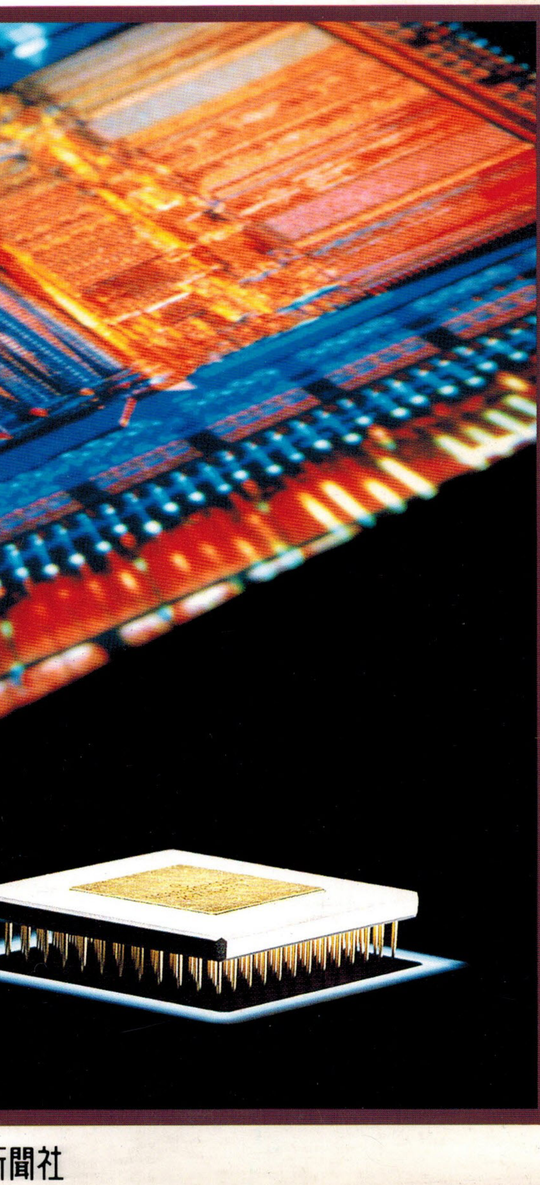
Computer Design

実践的システム設計のための――改訂版

68020ファミリ活用の実際

68030、68020、68010、68000、68008の機能詳解

辰野 功 著



電波新聞社

実践的システム設計のための68020ファミリ活用の実際
辰野 功 著
電波新聞社

実践的システム設計のための

68020ファミリ活用の実際

辰野 功 著

電波新聞社

序　文

マイクロコンピュータやパーソナルコンピュータの世界では、日進月歩の技術革新が積み上げられ、外部から一見して目立つ変化はなくても、内部では着実に技術が取込まれています。モトローラ社の32ビット・マイクロプロセッサ68020ファミリをベースにCPUシステムを解説した本書は、世の動きと連動して、執筆するうちに新しい項目の追加、重要なポイント、たとえばPMMU等解説の肉付けが加わり、ここ1年ほどを振り返ってみても話題に事欠きませんでした。

新しい半導体技術は、シリコン技術の進歩により、さらに新しい世界を開拓するパイオニアでありますが、とりわけ、台頭著しいディジタル・シグナル・プロセッサ、RISC型MPUなど、従来の汎用コンピュータと異なった定義をせざるを得ないニューフェイスの誕生により、さらにその芽を伸ばしつつあります。

こうした環境では、従来型のプロセッサの応用においてもシステム全般の理解が不可欠となり、同時に高度の機能を備えたMPU、周辺チップの効率的な活用能力がシステム設計者の決め手となるのはまちがいないところです。

本書は、数多くの諸先輩方の貴重なアドバイスをもとに、「活用するための知識」をまとめたものです。従って、入門書ではありません。設計の現場からお届けする実用書でありますが、8ビット系マイクロコンピュータのハードウェアの知識がある、またこれから16ビット、32ビットMPUへと設計対象を拡張したいとする読者諸兄の為に、基礎的な知識にも充分に配慮しました。

特に重点を置いたのは、MPUとOSの関係、ハードウェアとソフトウェアの接点の問題です。この問題は、本書の姉妹書である「68020ファミリ設計の実際」（電波新聞社刊）で、さらに具体的に解説しましたが、新しいハードウェア設計の際に必ず生ずる「設計からデバッグへ」の問題でもあります。

この視点から本書は、私の失敗とそのメインテナンスの紹介とも言え、現場からの現場の為の理論を集約した一種の「経験と理論が混在した書」という側面を持ちます。

本書の使用にあたっては、モトローラ社の各種ユーザーズ・マニュアル、本書「68020ファミリ活用の実際」、姉妹書「68020ファミリ設計の実際」の3冊が、「規格・動作の詳細」「それぞれの意義と概要」、「実際の設計／デバッグ作業」をカバーし、「規格－概念－実用」というシステム設計を3本の柱で支えている点を念頭に置いて読破いただければ幸です。とりわけ、「68020ファミリ設計の実際」と併読いただければ、効果が期待できるものと確信します。

本書が、他者の失敗を繰り返す愚を避け、効率的設計を目ざす読者諸兄の一助になれば幸です。

平成元年　初夏　　　　辰野　功

実践的システム設計のための

68020ファミリ活用の実際

目次　　CONTENTS

1 高級マイクロプロセッサへの進展とその位置付け

1-1 高級マイコンと従来機の相異点……10
1-2 機能の高級化……10
1-3 OS組込みを前提とする高級マイコンの設計……12

2 モトローラ68000ファミリの開発思想

2-1 開発時の基本設計のポイント……14

3 MC68000シリーズの特徴

3-1 基本アーキテクチャ……20
3-2 68000ファミリのタイプ……21
3-3 68000の命令実行……22
3-4 アドレス空間……23
3-5 MPUの製造技術……23
3-6 MPUの内部実行……24

4 一般的OSの使命と，その役割り

4-1 一般的なOS……26
4-2 OSの仕事……28
4-3 OSの実際……31
4-4 OSへの仕事の依頼……36

5 68000ファミリにおけるOSのサポート

5-1 スーパーバイザとOS……38
5-2 スーパーバイザへの移行……38
5-3 オルタネイト・ファンクション・コード機能……41
5-4 マルチOS……42
5-5 仮想記憶について……44
5-6 68000ファミリの仮想記憶サポート……46

6 68000ファミリの機能内容

6-1 データ・レジスタ……50
6-2 アドレス機能……50
6-3 68000ファミリのアドレス方法……53
6-4 SRについて……54

7 命令解説

7-1　68000ファミリの命令……60
7-2　論理，数値演算命令……62
7-3　68020のビット・フィールド命令について……65
7-4　システムの制御に関する命令……67
7-5　TRAP命令……67
7-6　マルチ・プロセッサに対する命令……68
7-7　デバッグ用の命令……69
7-8　68000ファミリのアドレス・モード……71
7-9　各アドレス・モード……72
7-10　アドレス・モードの注意……78
7-11　CALL命令……78

8 68000ファミリのアセンブラ書式

8-1　基本書式……84
8-2　アセンブラ中で使える記号……85
8-3　アドレス・モードのアセンブラ表記……87

9 メモリおよびハードウェア

9-1　メモリとメモリ空間……90
9-2　メモリと命令との関係……91
9-3　回路上から見たメモリ……92
9-4　MPUから見たメモリ……94
9-5　68000ファミリのアドレス表記法……94
9-6　ビット・フィールドとメモリ……96
9-7　パイプ(インストラクション・パイプ)……96
9-8　命令先読み機構……97
9-9　メモリのタイミング……98
9-10　メモリのアドレス・デコーダ……101
9-11　内蔵キャッシュ・メモリ……102
9-12　ダイナミック・バス・サイジングについて……105
9-13　68000ファミリと同期型I/O……107
9-14　メモリ・システム設計の盲点……108
9-15　MPUのリセット・スタート……109
9-16　MPUの実行速度……111
9-17　68000ファミリの実行時間……112

10 割込み処理

10-1　例外処理の概要……118
10-2　例外処理への移行手順……122
10-3　例外処理とバス・エラー……124
10-4　バス・エラーの入力に対するMPUの例外処理以外の対応……125
10-5　アドレス・エラー……126
10-6　特権命令違反……127

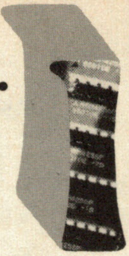
10-7 外部割込み手順の詳細……127
10-8 外部割込み処理回路……129
10-9 例外処理時のスタック・フレーム……130
10-10 マルチCPUのサポート……131
10-11 バス・アービトレイション……135

11 コ・プロセッサ
11-1 コ・プロセッサに対する知識……138
11-2 コ・プロセッサとの通信……138
11-3 コ・プロセッサ・インターフェース……139
11-4 コ・プロセッサと外部割込み……140
11-5 コ・プロセッサのセーブ/リストア……141
11-6 コ・プロセッサ命令……142
11-7 コ・プロセッサの命令の実行……142
11-8 コ・プロセッサの実行の中断，再開……143
11-9 コ・プロセッサ発生の例外処理……144
11-10 コ・プロセッサとの通信のデバッグ……144
11-11 MC68881/2……145
11-12 MC68851……149

12 MMUの役割りと機能解説
12-1 MMUとは何か？……152
12-2 MMUのその他の機能……153
12-3 MMUの種類……154
12-4 MMUによるアドレスの変換法……155
12-5 MC68451……156
12-6 PMMU68851……157

13 MC68030の機能解説
13-1 MC68030の概要……172
13-2 PMMU68851との違い……172
13-3 MMUの性能……173
13-4 68030のキャッシュ・メモリ……175
13-5 バースト・モード……179
13-6 キャッシュ自体の構造……181
13-7 バスの動作……182

14 68000/10システムの68020への変更 68020システムの68030への変更
14-1 ハードウェアの全体構成……186
14-2 システムの性能……186
14-3 ソフトウェアの互換性……186
14-4 ハードウェア上の問題点……187
14-5 個別機能の説明……187
14-6 回路上の説明……190
14-7 アダプタに対応するソフトウェア……192
14-8 68030の68020システムへの適用……193



第1章

高級マイクロプロセッサへの進展と，その位置付け

1-1 高級マイコンと従来機の相異点

集積回路技術の向上によって，従来ではミニコンや中型機のものと思われていた本格的なコンピュータを，数チップで構成する事が可能となった。

そこで，4ビット，8ビットまたはその延長の単純な16ビット・マイコンとは違ったコンピュータが登場しつつある。とりあえず，従来の4，8ビット・マイコンと高級マイコンの違いについて簡単に触れておきたい。

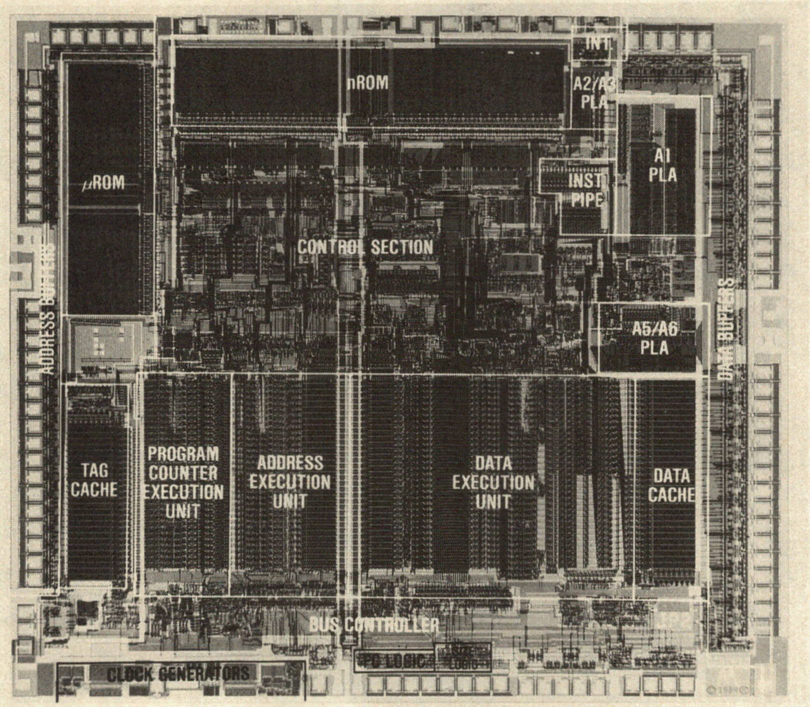
今まで，MPUについて，4，8，16ビット・マイコンというように数字で区別してきたが，この数字は一般にマイコンに接続されるデータ用のメモリの幅を表わしている。8ビット以上のマイコンではほとんどの場合，データと命令の幅は同じだが，下位の4ビットの場合，データは4ビット，命令は8ビットといったものも存在するし，また上位のものではデータは32ビット，命令は16ビットというものもある。また，近頃流行しはじめているディジタル・シグナル・プロセッサは，データと命令の幅は大きく異なり，幅の広いデータを扱える様になっているものが多い。

データの幅が広いという事は何を意味するかというと，一回に扱う事のできるデータの量が大きくなっている事である。例えば，同じ半導体の製造プロセスを使って得られる素子の速度，つまりロジックの速度はほぼ同じである。したがって，コンピュータの処理能力を上げようとすれば，一時に大量のデータを処理できる方が有利な訳である。同じ半導体プロセスで作られた8ビットと16ビットのマイコンならば，16ビットの方が8ビットの2倍速い事になる。もっとも，実際には，1ビット単位のデータを扱う場合もあるので，この時には速くならない。従って，メモリの幅は，応用により広くても役に立たない場合もある。4ビットや8ビットの下位MPUは，逆にこの様なビット単位の処理に適しており，16／32ビットのマイコンは，幅の広い大量のデータを処理するのに適していると言える。また，16ビット・マイコンの内部の演算機構も上等に作られており，乗算，除算等の8ビットにない機能を有していれば，その分，速くなる。更に命令自体も，8ビットより16ビットの方が多種多様に作れるため，高い機能を16ビット・マイコンに与える事ができる。つまり 2^8=256の基本パターンしかないものと，2^{16}=65536の基本パターンを持てるものの差ということになる。

この様に，高級指向のマイコンのメモリの幅は広ければ広い程良いが，実際にはIC化する時の物理的な制限がある。例えば，ICのパッケージのピン数，1マイコンに使えるシリコン・チップの面積等々から，それほど大きくはできない。汎用マイコンでは，おそらくデータの幅は32ビットが限度と思われる。IBM，DEC等のミニコンも32ビットのものが大勢を占め，現在機能の高級化は，付属の数値演算専用回路や並列化によって行なおうとしている。もちろん，ごく短時間に，大量のデータの繰返し処理のみを行なうという専用化されたプロセッサ，例えばディジタル・シグナル・プロセッサやRISCでは，更にデータの幅は広がると思われる。特にこの種のLSIは，シリコン・コンパイラによってカスタム化が容易になってきているため，ますますデータの幅を広げてゆく傾向を示している。

1-2 機能の高級化

データ・バスの幅が広がり命令の種類が豊かになると，各命令が高級化する。これは命令自体の幅の拡張によって，様々な細かい処理，条件の指定ができる様になるためで，4，8ビット・マイコンでは3～4命令かかっていたのと同じ処理を1命令で行なう事ができる様になる。一般のマイコンはノイマン型のコンピュータだが，命令，データとも同じデータ・バスを通して処理されるため，この命令を読む時間は確実に減少する。もちろん，IC自体の速度が同じなら(実際には後から作られた分だけ新しい製造技術を使っているので速い)一つの命令に要する時間は長くなるが，外部のメモリを読み出す時間に比べ，内部処理の方が段違いに早く，全体では格段に速くなる。



▲32ビッドMPUモトローラMC68020

現在の高級指向のマイコンは，ほとんど全て内部がマイクロ・プログラミングという技法により(例外：Ｚ8000)，内部的にもソフト化されている。外部のメモリから読む１命令は，内部のマイクロ・プログラミングの特定の何ステップかの処理を行なわせる事になる。ところが，どんな集積回路にも言える事だが，ＩＣチップ上の信号が伝わる速度とチップ外に出る速度を比べると，およそ10倍程も内部信号の方が速い。したがって，内部処理の比率が高ければ高い程，高速化する事になる。現在の高級ＭＰＵのチップを見ると，メモリのチップかと思われる程チップ面積上のメモリ部分の比率が高くなっている。そして，これらのメモリを用いての内部のソフトウェア化により，各命令の柔軟さが増し，ファミリの拡張が容易となる。この傾向は高級マイコン全般にわたっており，ＤＥＣ社のLSI－11，㈱東芝の16ビット・マイコンは，このマイクロ・プログラミングＲＯＭを外付とし，ＩＣとしての製造技術上の違いを避けて高機能化，高集積化を目指している。

更にマイクロ・プログラミング制御，つまり内部のソフトウェア実行によって行なわれる複雑な処理に対応できる複雑高度な演算処理部は，当然高級化され，８ビットとは比較にならない速度と能力を持つに至っている。

また，高級マイコンは扱うデータの量が当然大きくなる。そこで，扱えるメモリの量を大きくするためアドレス線の数を増やしていく。代表的な８ビット・マイコンは，データ／命令８ビット，アドレスは16ビットというスタイルが多かったのだが，16ビット・マイコンでは，アドレスが20，24，32と増やされている。当然扱えるメモリの量も，８ビット・マイコンが64Kバイト＝ワードであったが，16ビットでは少ないものでも２Ｍバイト，大きいものでは４Ｇバイトという様に拡大されている。

メモリがこれだけ大きくなり，実際には数メガ・バイトのメモリを扱う事となるとマイコンがメモリ

を全て書き直していたのでは，とても高い効率は望めない。そこで，直接メモリ―メモリ，またはメモリ―入出力間のデータ転送を行なわせる外部の補助機構が欲しくなる。これがダイレクト・メモリ・アクセス・コントローラ（DMAC）と呼ばれる機構で高級マイコンにはシリーズに必ずDMACを持っているし，ＭＰＵ内に組込んでいる例もある。

また，使えるメモリが大きくなり，メモリ上で，各種のプログラムがマルチタスクで動作する様になると，プログラム作成上の都合，また実行用の命令コードの効率的な発生のために，ＭＰＵにとって同じアドレス・エリアを使う事が多く起きる。そこでアドレスとメモリの関係を調節し，同時にメモリ上に存在するプログラム間の干渉を避けて，アドレスの番地とメモリの番地を変換し，また命令コードへの書込みを避ける等の機構が必要になってくる。この様な機構をメモリ・マネージメント・ユニット（ＭＭＵ）といい，内蔵するか，容易に外付けできる様にするのが一般的であり，既にこの機能の一部または全部をＭＰＵ上に組込んだＭＰＵも存在する。

先に演算部の高級化について言及したが，一般に汎用コンピュータの演算部の高級化は，汎用であるという条件下で行なわれるため，そのＭＰＵをユーザーの20%程度しか使わないような機構で複雑なものはむしろＭＰＵ上に搭載せず，外部に拡張できる様にする場合がある。このような命令＝能力拡張機構をコ・プロセッサといい，高級ＭＰＵでは必須条件の一つとなっている。その代表的なものとして数値演算コ・プロセッサ（8087，68881等）がある。

このようにMMU，ＤＭＡＣ，コ・プロセッサ等の外部機構が充実すると，ミニコンでは広く行なわれている仮想マシンを作る事ができるようになり，それだけＭＰＵの内部機構がミニコン並みに充実されてきたということになる。

仮想マシンとは，MMUと磁気ディスクによってＭＰＵに接がれているメモリを，磁気ディスク上のデータと入れ替え，メモリが無限にあるかのごとくに見せかけるテクニックで，メモリは磁気ディスク上のデータのコピーとなる。ＤＭＡＣにより磁気ディスク上のデータとメモリのデータを次々と入れ替えＭＰＵの命令実行は継続される事を可能とするものである。詳細は後章にゆずり，ここでは概念だけにとどめる。

1-3　ＯＳ組込みを前提とする高級マイコンの設計

さて，従来のプロセッサと新しい16／32ビット高級プロセッサの大きな違いは，マイコンの計画段階から，何らかのオペレーティング・システムの組込みを前提として設計されていることである。

ここでいうオペレーティング・システム（ＯＳ）は，マイコン・システム自体を管理するプログラムの事で，入出力素子，磁気ディスク，フロッピーディスク，メモリ管理ユニットやＤＭＡＣ，各種割込みの管理等々，それ自体は特定の仕事に関与する訳ではないが，システムを使いやすく，効率よく運営させるためのソフトウェア環境を提供する目的で作られたプログラムである。有名なものに米国ベル研究所のUNIXがある。16／32ビットの高級ＭＰＵはこの様なＯＳを組込んだシステムを作る場合，ＯＳが欲求する回路的な要求を効率良く実現できる様にＯＳサポート回路を組込んでいるのが特徴である。

更にスピードの向上のため，命令先読み機構の充実，キャッシュ・メモリの組込み等が行なわれ，動作クロックは従来のものより２倍～５倍の速さとなってきている。この結果，総合的に見て従来のミニコンに勝るとも劣らない本格的なシステムを組む事ができる。

また場合によっては，ある時点における処理の要求する極度に高い性能を満たす為に，この高級ＭＰＵを比較的単純な処理に用いるという瞬間最大風速を求める場合の使い方の様に従来の８ビット機と同様の使い方もできるのが現在の16／32ビット高級マイコンと言えるだろう。

筆者注）以下，ＭＰＵ（マイクロ・プロセッシング・ユニット）とはマイコンのチップ自体を指し，ＣＰＵ（セントラル・プロセッシング・ユニット）とは，処理を行なえる機能ユニット，つまり，ＭＰＵ＋ＲＯＭ＋ＲＡＭ＋(Ｉ／Ｏ)を指す。

第2章

モトローラ68000ファミリの開発思想

2-1　開発時の基本設計のポイント

68000は，16ビットのデータ・バスをサポートする本格的なマイクロプロセッサとして開発されたが開発時点において，幾つかの重要な選択がなされている。それを理解する事は，このファミリのより深い理解への一助となるものと思う。

1〕 下位の８ビット系のＭＰＵと何らかの互換性を持たせるか？

6800／09ファミリとの互換性は持たせない。これは８ビットの下位のアーキテクチャの制限によってこの16／32ビットＭＰＵの設計に無用の制限をかけない様にするためである。

ただし，現在の入出力用周辺ＩＣ(68系)とのハードウェア上の接続はできる様にして，広く８ビット系の入出力を利用できる様にはする。

2〕 データ処理の単位について

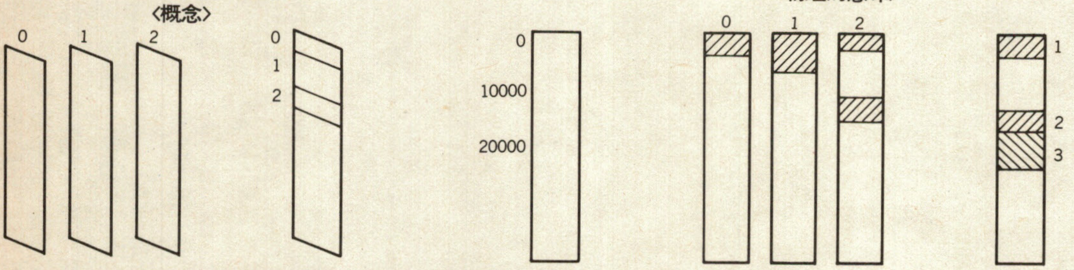
まず，68000ファミリでは，基本的にデータの処理単位をバイト(本書では大文字Ｂ)とする事となり，データ・バスの幅によらずに，メモリ空間のアドレス単位は全てこのバイトを単位とする事となった。これにより，ファミリ内でのプログラムによるアドレッシングの完全な互換性が保たれる。

そして，アドレスの処理単位は16または32ビットであり，リニアなメモリ空間のサポートが行なえる様にする事となった。このために，内蔵のレジスタはファミリ全てに対し32ビット長とする事になる。そして理想的なＭＰＵとするには，全ての汎用レジスタに何の差もない事が望ましいが，スタック・オペレーションやＡＬＵ（論理，数値演算ユニット）との協調関係から，一般の汎用レジスタ（アキュムレータ）は８本とし，バイトの処理，論理演算，高度の数値演算をサポートし，これらとは別に，アドレス・レジスタとして（これらは，ほぼＩＮＤＥＸレジスタに同じであるが）スタック・オペレーション，自動インクリメントーデクリメントのサポート等を行なわせる32ビット・レジスタを８＋１本用意する事とした（ここでの＋１は，後述するＳ／Ｕの切替に伴って自動的に切替わるシステム・スタックである)。

3〕 アドレス線を何本にするか？

最も簡単なのは16本であり，これは68000のデータ・バスの幅とも一致するため，ハード・ソフト上も簡単である。しかし，将来，上位のＭＰＵを開発する場合には，上位のアドレスは必らず必要となりまたデータ・バスの幅も必ず拡張されるであろう事

図2-1-1　ページングとセグメンテーション



●ページング
システムの有する実メモリを一定の小部分に切出し，これを１ページとして各タスクの専用空間に与えてゆく。

●セグメンテーション
実メモリをシステムの要求により，様々な長さで切出し各タスクに与えてゆく。

●リニヤ空間
特定の分割のないメモリ空間

●ページング
上位アドレスの拡張により，MPU固有のリニア空間を複数作り，その各々に何ページかのメモリを与える。

●セグメンテーション
必要量のメモリ（セグメント）をタスク番号に対し，要求により複数与えて，各タスクが独自の空間を持つ様に仮想する。物理的には1空間

が想定された。その際，ページングまたはセグメンテーション（両方とも外部の主記憶を何らかの方法で分割して64Kワードに納まる様にする方式。図2-1-1参照）の処理が必要になり，そうなると，ハード／ソフトとも複雑化して不利になる。そのため，ピン数の増大，チップ上の集積度の高まりがあっても単一アドレス空間方式にて，最大32ビットをサポートし，性能的に上位ミニコンをめざすものとする。ただし，ＭＰＵのチップ上の制限やパッケージに標準品を使うために，アドレスの上位の何ビットかは出力できなくても仕方のないものとする。

4］　ＭＭＵ機能を内蔵させるか？

2］の決定に伴ない，チップ上にMMUを乗せない事が同時に決定された。これは，ＣＰＵは充分に広い単一のアドレス空間を持ち，かつ，そのＣＰＵが容易にポジション・インディペンデントのソフトウェアの製作を許すものなら，システムのハードウェアの負担を軽くしたままで，マルチ・ユーザー，マルチ・タスクの環境を作る事が可能と判断されたためでもあった。

ただし，アドレスの上位拡張の意味も含め，ファンクション・コード出力を設けて，簡単なMMU機能を実現できる様にし，MMU素子を外部に接続する際には，簡単に本格的ＯＳをサポートできるものが接続できる様にした。

そして将来，このMMUを必要とするユーザーが多く出てくるであろうが，ＭＰＵを単機能として使う場合には不用であり，価格を押し上げることになるとの理由により，必然的に外付けのディバイスとして，とりあえずは別に開発される事となった。

5］　ＤＭＡ等の機能をＣＰＵに持たせるか？

3］と同様の理由により，ＤＭＡ，コ・プロセッサ等のＣＰＵ機能を強化する機能も，全て外付ディバイスとする事になった。

これにより，68000は汎用のＭＰＵとしての理想になるべく近づける様に，また現在のミニコンや中型機のＣＰＵで使われているテクノロジーを取り入れ応用できる様にし，ＭＰＵとしてバランスのとれたものとなる様に努力を集中する事ができた。

6］　仮想マシンをどうサポートするか？

仮想マシンのサポートに対しては，従来より理論的には知られていても実現される事の少なかった命令継続法を採用した。これは，ＭＰＵの内部の実行をハードウェアに頼らず，マイクロ・プログラミングを採用した最大の効果でもある。つまり，あるバス・サイクルの中断は，マイクロ実行に対する割込みとして扱われている。

内部的にマイクロ・プログラミングを採用した特徴は他にも出ている。上位機種に対しての上位命令の追加，アルゴリズムの変更改良による処理速度の向上等である。

また，この点で量産性にも良い結果がもたらされている。これはチップの面積の大半がメモリ（ＲＡＭとROM）で占められているという，ＣＰＵとしては例のない構成となっているため，最も効率的に製造技術の向上をとり入れることができたからである。

7］　データ・レジスタの幅と数をどうするか？

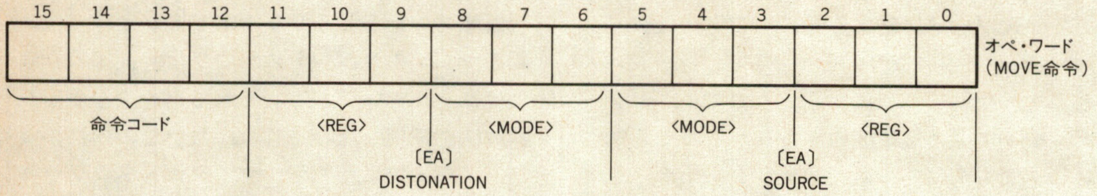
最大の問題として内部のデータ処理長が問題にされた。アドレスは32ビット処理と決まっていたため内部レジスタの互換の点からも，先進のプロセッサを目指すという点からも，データを扱うレジスタ長は32ビットとする事が最も望ましく，32とする事に決まった。また命令ワード上の制限から，レジスタは全部で16個しか使えないことが決定された。

これは命令ワードの構成上（オペワードの構成の項参照），レジスタはアドレス／データの２種を，それぞれ３ビットで０～７の番号を割り当て，各８個計16となるのが合理的と決定されたからである。

8］　オペ・ワード（命令のビット・フォーマット）の構成

レジスタが16個になってしまう理由は68000ファミリのオペ・ワードにある。68020/30ではこの後にポスト・ワードの付くものもあるが，68000ファミリのオペ・ワードは16ビットで最小一つの命令を構成しその上位４ビットがオペ・コード，残り12ビットを

図2-1-2 代表的な68000系MPUの命令ワードのビット配分



6ビットずつに分け，その各6ビットが命令のソース側とディストネーション側を表わし，6ビットを3ビットずつに分け，各3ビットがアドレス・モードとレジスタ番号を表わすというのが，基本形と決まったからである(図2-1-2，図2-1-3参照)。もちろん条件分岐命令「Bcc」系の命令の様に，ソース，ディストネーションがないため，12ビットを4と8に分け，4が判定条件，8が相対位置を示すものもある。

この最上位4ビットをオペ・コードと称するが，これらの示す意味を，**表2-1-1**に示す。

この内，例外処理の「LINE ＄Aエミュレータ」の＄Aは，この4ビット・コードの空きエリアの16進表示からきている(例外処理の項参照)。

ソース／ディストネーションの指定方法は，この〈モード〉＊〈レジスタ番号〉の積となって，広く様々な方法をサポートする。ただし，命令によって使えないモードもあるが，それは後述する(68000の命令参照)。この〈モード＊レジスタ〉はアドレッシング・モード及びその影響する所と深く関わっている。これをまとめたのが**表2-1-2**である。

9〕 演算の幅

5〕と同様に，ALU(演算ユニット)に直接接続されるレジスタ群，データ・レジスタ8個と，アドレス・ユニットに接続されるレジスタ群，アドレス・レジスタ8個共32ビットで，ALU自体も理想的には32ビットで構成したいのだが，68000／10／08版では，ALUの幅は16ビットである。その理由は設計の複雑さ，集積度の異常な高まり，内部論理素子のスピード等の制限であった。これは，68020になって，データ・バスと同時に32ビットに拡張された。

この拡張は相対アドレスの計算範囲にもあり，PC相対でのPCの値の変化は，68000／10版ではオフセットは16ビットであったが，68020では32ビットが可能になっている。

10〕 条件フラグをどうするか？

8ビットの時代から引き継がれたフラグ(キャリー，ゼロ等)の自動セットも当然サポートされた。

11〕 命令読取り機構の改善をどうするか？

命令のデコードはパイプライン化され，命令の読取りは実行効率を上げるためにも，プリフェッチ(命令の先読み)を行なわせる事になった。

12〕 高級言語のサポートをどうするか？

命令は高級言語をサポートしやすい様にLINK／

表2-1-1 オペレーション・コード・マップ

ビット15～12	操作
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/Scc/DBcc/TRAPcc
0110	Bcc/BSR/BRA
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned,Reserved)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate/Bit Field
1111	Coprocessor Interface

UNLK等，また強力なデータ・スタック機能等のサポートが行なわれた。この点では，むしろアドレス・モードの拡張により，ＰＣ相対テーブル・インデックス等の機能拡張の方に重点が置かれた。

更に８ビットの入出力をサポートするために，１バイトおきのデータ転送ができる「MOVEP」の様な命令もサポートされた。

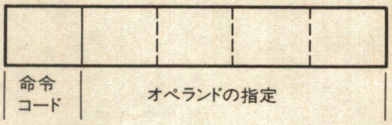
13］割込み処理（内部エラー）をどうするか？

内部からの割込みの処理については，外部からの割込みと内部の発生をほぼ同様に扱う事に決まり，８ビットと同様ベクタ・テーブルを採用した。しかし，そのベース・ポイントは，6800／09系のアドレスの最高位でなく最下位に移された。

これは，このテーブルを複数持つ様になった時，または拡張した時の処理の容易さのためである。

内部発生の割込みは，ＯＳを組込んだ時に充分に様々なＯＳテクニックをサポートできる様にする為に，他のプロセッサに比べ大幅に拡張された。外部

図2-1-3　68000のオペ・ワード



からの割込みは，外部からベクタを指定する方法と内部の特定のベクタを利用する方法の両方が使える様になった。

14］メモリ・アクセスをどの様にするか？

メモリのアクセスに対し，システム・クロックに同期させていた８ビットの手法を捨て，ハンド・シェーク法をとる事になり，システムの中に各種のスピードのメモリが混在する事を容易に許す事になった。更に，８ビット系のＩ／Ｏをサポートするため68000／08／10では，外部からの指示により，6800式の同期型のサイクルも利用できる様にした。

表2-1-2　実効アドレスの〈モード〉と〈レジスタ〉のアセンブラ表記，及び各アドレス・モードの動作による分類

アドレッシング・モード	モード	レジスタ	データ	メモリ	制御	可変	アセンブラ・シンタックス
Data Register Direct	000	reg. no.	×	–	×	×	Dn
Address Register Direct	001	reg. no.	–	–	–	×	An
Address Register Indirect	010	reg. no.	×	×	×	×	(An)
Address Register Indirect With Postincrement	011	reg. no.	×	×	–	×	(An) +
Address Register Indirect With Predecrement	100	reg. no.	×	×	–	×	−(An)
Address Register Indirect With Displacement	101	reg. no.	×	×	×	×	(d_{16},An)
Address Register Indirect with Index (8-8it Displacement)	110	reg. no.	×	×	×	×	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	110	reg. no.	×	×	×	×	(bd,An,Xn)
Memory Indirect Post-Indexed	110	reg. no.	×	×	×	×	([bd,Anl,Xn,od)
Memory Indirect Pre-Indexed	110	reg. no.	×	×	×	×	([bd,PC,Xn],od)
Absolute Short	111	001	×	×	×	×	(xxx).W
Absolute Long	111	001	×	×	×	×	(xxx).L
Program Counter Indirect with Displacement	111	101	×	×	×	–	(d16.PC)
Program Counter Indirect with Index (8-Bit Displacement)	111	011	×	× ×	×	–	(d8,PC,Xn)
Program Counter Indirect with Index (Base Displacement)	111	011	×	× ×	×	–	(bd,PC,Xn)
PC Memory Indirect Post-Indexed	111	011	×	×	×	–	([bd,PC]Xn,od)
PC Memory Indirect Pre-Indexed	111	011	×	×	×	–	([bd,PC,Xn],od)
Immediate	111	100	×	×	–	–	# <data>

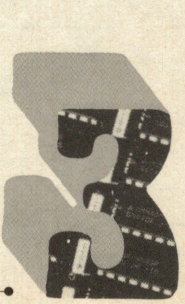
15〕マルチ・プロセッサ・サポートをどうするか？

マルチ・プロセッサのサポートは，あるメモリでのフラグの有無のチェックとその結果による自動フラグ・セットを行なう「ＴＡＳ」１命令をサポートし，セマフォがエラーなく構成できる様にする事のみとなった。これで，二つ以上のＣＰＵは互いのフラグのやり取りをセマフォを通じて正しく実行できる。

この点は，68020では更に「ＣＡＳ命令」が追加された。これは，あるレジスタの内容とセマフォの内容を比較して，一致した場合には別のレジスタの内容をそのセマフォに書き，不一致ならセマフォの内容を比較用のレジスタに読み，セマフォの内容をチェックできるものである。

第3章

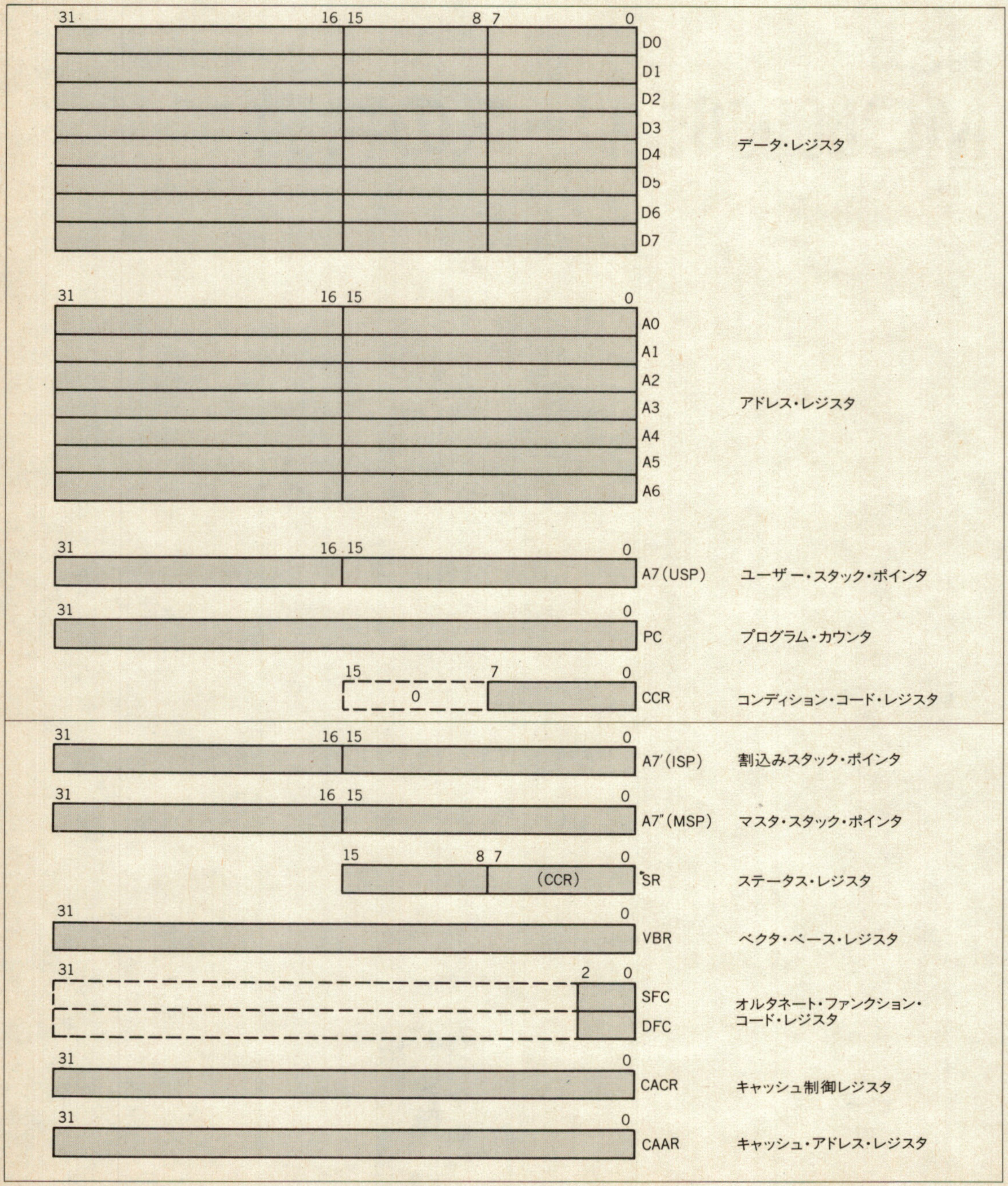
MC68000シリーズの特徴



3-1　基本アーキテクチャ

ＭＣ68000シリーズには，データ・バスが32ビット，データ・キャッシュやMMUまで内蔵した68030から８ビットの68008があり，サポートしているデータ・バス・サイズは８／16／32，またアドレス・バス・サイズも20／23／30／32と各種あるが，シリー

図3-1-1　プログラミング・モデル



ズのベースとなった68000の基本内部構造は同一であって，ほとんどのソフトウェアが互換であるという統一された設計思想を基礎に作られていることが大きな特徴である。もちろん，ただ単に統一されているだけでは特徴とは言えないが，68000のシリーズは，その内部構造が全て32ビットで作られている点が大きな特徴となっている。

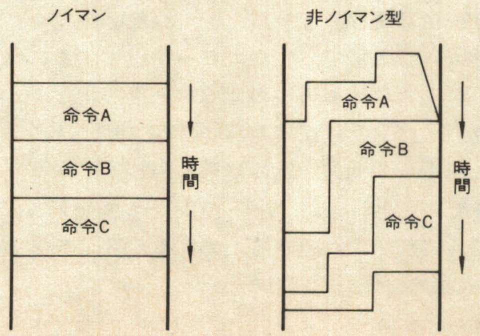
68000シリーズ各タイプの差は，基本的にはその内部構造をどれだけ外部に反映させたか，更にもう一つの点は，仮想マシンを自己でサポートできる様に改良されているか，の点についてである。

更に内部構成の単純さも，魅力の一つであろう。68000シリーズのソフトウェアが主に扱う32ビットのレジスタは，特殊な場合に用いられるもの以外には３種類しかない。

●主にデータの処理を行なう＝データ・レジスタ（Ｄ０～Ｄ７）８本
●主にアドレスの処理を行なう＝アドレス・レジスタ（Ａ０～Ａ６，ＵＳＰ，ＳＳＰ）９本又は10本
●ＣＰＵの状況を示す＝ステータス・レジスタ（このＳＲは16ビットまたはコンディション・コード・レジスタ）ＳＲまたはＣＣＲ
（注）ＣＣＲはＳＲの下位８ビット

もちろん68010以上のシリーズには，この他にも割込みベクタのテーブル位置を示すベクタ・ベース・レジスタや，モードの異なるアドレス空間をアクセスするためのオルタネイト・ファンクション・コード・レジスタが追加され，68020になると更に割込み専用のスタック（ＩＳＰ）が追加される。

図3-2-1　ノイマン型と非ノイマン型



3-2　68000ファミリ・ＭＰＵのタイプ

68000ファミリは全て通常のノイマン型コンピュータである。

つまり，メモリから命令を読み取り，その命令の指示によって演算を行なうコンピュータであり，一時に一つの命令のみを実行し，一つの命令が実行し終わると，次の命令を実行するといったタイプのコンピュータである。従って，いくつかの処理（並列処理）を行なう非ノイマン型マシンではない（**図3-2-1**参照）。

また，扱うデータは命令のオペランドによって指示されるが，普通メモリ上のデータを入力とし，出力もメモリ上に行なう。特に68000ファミリは，入出力素子もメモリとして扱うメモリ・マップド・Ｉ／Ｏを採用しているので（**図3-2-2**参照），データは全てメモリ上にあり，これを処理すると考える事ができる。

従って，68000ファミリの命令には，入出力専用空間をサポートする命令はない。同様に，Ｉ／Ｏ空間を指示する専用のアドレス線，またはこれに類するハードウェアもなく，ユーザーは任意のアドレスに入出力素子を置くことができる。但し，一般のコンピュータの制限と同様，メモリ上のデータはそのままでは演算はできず，一部または全部をプロセッサ内部に取込む必要がある。68000ファミリでは，この様なデータを主に扱うプロセッサ内の演算ユニット（ＡＬＵ）と，密接に結びついている演算用の内部レジスタがデータ・レジスタである（**図3-2-3**参照）。

したがって，乗除算の様な高度の処理は，このデータ・レジスタ間のみで行なわれる。逆に加算やデ

図3-2-2　メモリ・マップドI／O

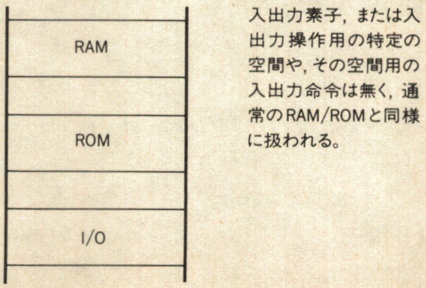
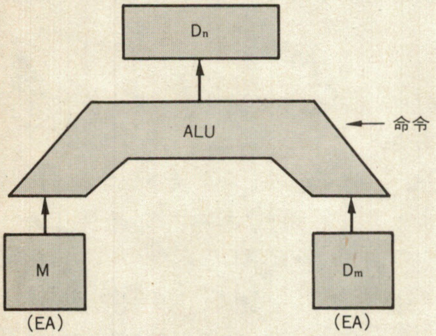


図3-2-3　ALUと命令の実行



一般の命令はALU（論理演算ユニット）によって処理される。この時，二つの入力データの各々が〔EA〕によって指定され，ALUには命令が与えられる。

ータの移動の様な簡単な処理は，メモリ—メモリ間でも行なう事もできる。プログラム・ループの数を指定したり，大小比較をする様な場合は，やはりこのデータ・レジスタが使われる。

この様なレジスタを，ミニコンの世界では汎用レジスタ（アキュムレータ）と言い，ミニコンではALUと共に設計の最重要点とされるものである。

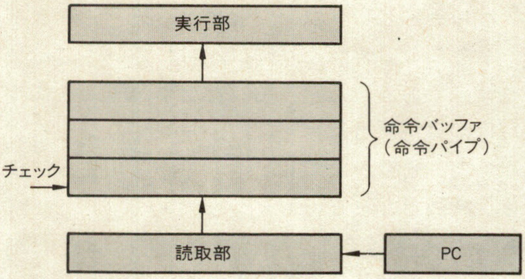
8ビットのマイコンではこの様な完全な汎用レジスタは，回路上の制限により構成しづらいため，演算専用，計数専用等に機能分けしているものが多いが，本来は用途を特定されずに使えるものである。

この他に，MPU内には，アドレス作業を主として行なうアドレス・レジスタ，MPUの機能の制御と実行によりセットされ，又，実行のコントロールに用いられるフラグのレジスタを有する。更に，上位機種では様々な内部レジスタが追加されている。これを図3-1-1に示す。

3-3　68000の命令実行

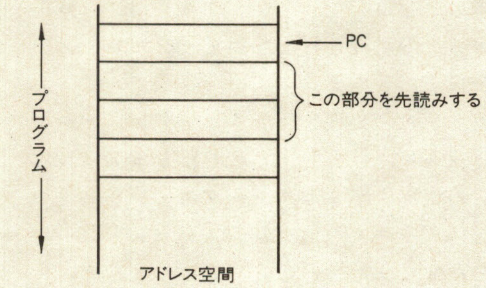
先に68000ファミリは，ノイマン型コンピュータであり並列処理は行なわないと記述した。確かに各命令レベルでみるとその通りだが，MPUの持っている機能から言えば，必ずしも正しくない。と言うのは，68000ファミリは，命令の実行部とメモリからの命令読取り機構，MPUのバス・コントロール部がある程度独立した動作を行なうため，ある命令を実

図3-3-1　命令読取り部・バッファの独立した動作



命令読取部は，命令バッファに空が出来るとPCのアドレスにより，次のプログラムを読みバッファにつめる。
バッファでは，不正命令等のチェックが入り，適時，実行部（ALU部分）に与えられる。
イミディエイト・アドレス・モードでは，このバッファにあるオペランドの内容がそのまま使われる。

図3-3-2　エラー発生の留意点



行部が演算中に命令読取り部は次の命令やオペランドを読み次の命令の実行に備えているし，この段階で命令コード自体のエラー・チェックも行なわれる。そのため未定義の命令を実行してしまう様な事態は生じない（図3-3-1参照）。

68020では，もし未定義の命令を命令読取り部が読んだ場合に，実行ユニットの所までその命令がこなければ，内部でのエラー処理が行なわれない。68000では，命令読取り部が読んで実行部と接なぐバッファに入った段階で，すぐエラーが発生する。つまりプログラム自体の最後のオペワードの次と，その次のワードはMPUが読む可能性があり，そこに未定義の命令や，内部割込み処理を引き起こすコードが入っていると，そのコードが実行ユニットに入らなくても内部処理が起きるため注意が必要である（図3-3-2参照）。

同様に，オペランド処理のためのメモリ・コント

ローラが，この命令の先読み（プリ・フェッチと言う）と競合しない様に，オペランド・アクセスを行なうので，各メモリ・サイクルは，命令の実行順序通りとは行かない。

3-4　アドレス空間

メモリをワードまたはバイトの幅を持つものとして，図柄上，横方向に幅をとり，垂直方向にアドレス線の組合わせで示される量を表わすと，メモリのビットを縦／横に配列した２次元の平面空間を表わしている様に見える（図3-4-1）。

この様にして，ＭＰＵの処理できるメモリの内の特定のメモリの位置を指定するが，これを表わす２次元の空間全体をメモリ空間と称する。

実行すべきプログラムや，演算すべきデータは，全てメモリ空間の中にある。これらは，大量にあるので，ＭＰＵは必要に応じて特定のメモリを指定して，そこを読出したり書込んだりする。このメモリの特定位置の指定は，ＭＰＵのアドレス線を通じて行なう。このアドレス線がもし１本なら，その線のレベルが０の時と１の時とで，２ヶ所が指定できることになる。同様に，２本なら１と０の組合わせにより４ヶ所……という様に，２のＮ乗（Ｎがアドレス線の数）個所のアドレス空間を構成できる。68000ファミリでは，全てのアドレス空間を単調増加の一つの空間として扱える（3-4-1参照）。

また，プロセッサが一時に扱うデータの量は，メモリのビット幅による。これは，プロセッサ自体のデータ入出力線の数を言い，プロセッサが８ビットであるとか16ビットと言うのは，ここの幅を指している。これは，アドレス線によって指定された，ある１ヶ所のメモリのビット数とも言える。68000では，これが16ビット，68020では32ビットである。

68000は２バイト，68020では４バイトとなり，68000ファミリはメモリはバイト＝８ビット単位の量で数える。68000のアドレス線は，23本あるため8,388,608ヶ所のアドレス空間の指定ができる。略して８Ｍワード。ここではワード＝バスの幅なので16ビット＝２バイト，つまり８Ｍワード＝16Ｍバイト

図3-4-1　A2, A1, A0の3ビット・アドレスによる8ワードの空間

A2	A1	A0	← ワード →
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

ワードの幅は各CPUによる。
この空間は8×（ワードを構成するビット数）のビットで構成される2次元平面空間である

図3-5-1　メモリ空間内での位置の指定

値	A0	← ワード →
0	0	
1	1	

値	A1	A0	← ワード →
0	0	0	
1	0	1	
2	1	0	
3	1	1	

アドレスによって作られる空間の幅は，アドレス線の数の2^n×（ワード幅）となる。また，各アドレスを区別する為，各パターンを2進カウンタの表記値と一致させた数値として表す。

のアドレス空間があるという言い方をする（図3-5-1参照）。このメモリを各命令のオペランドで指示する方法をアドレス・モードと呼び，68000／08／10では14種，68020では18種となっている。また，各アドレス・モードで指定されるアドレスを実行アドレス＝エフェクティブ・アドレス（ＥＡ）と称す。これは実行といっても，そこに対してデータ処理を実行するという意味で，読出しも書込みもあり得るし，またアドレス空間上のメモリでなく，ＭＰＵ内のレジスタの場合もある。

3-5　ＭＰＵの製造技術

68000ファミリは，そのスタートである68000ＭＰＵの時代には，高密度化されたＮＭＯＳのディバイスとして開発された。この68000自体の販売開始は1979年のことであり，これから判る様にかなり古い

技術の産物である。日本にも1979年の終りごろには第1ロットが入ってきている。

このNMOS（2.5μm）の製品のシリーズは，68008，68010，68012まで引きつがれている。このシリーズの製品の特徴はリセットを与えるべき時間にある。これらの製品のいずれもが200mS以上となっているが，これは，チップ内部のコクロフト（スイッチングによる負電圧発生器）が安定に電源を供給できる様になるために必要な時間であり，チップ内にあるデータ・レジスタ，アドレス・レジスタ，等々，全てのレジスタは，ダイナミックな素子として作られている（ダイナミック・メモリの各メモリ・セルと同じ。ただしリフレッシュは完全に内部で行なわれ，外部からは全くこれを知る事はない）。

ちなみに，68000のトランジスタ数は68000トランジスタと称されている。ここまでの各シリーズは，このトランジスタ数の大幅な変化はない（<10万トランジスタ）。

これが，68020になると，大きく様変わりする。まず，製造技術がCMOSになって熱の発生量＝消費電力が大幅に減った。ただしレジスタ類のダイナミック動作に変わりはない。これは，68000と同じである事が，クロックに最低動作周波数があることからもわかる。そしてトランジスタ数は約24万トランジスタとなり，68000の約3倍となっている。この中には，約60ビットの長さを持つ命令キャッシュ・メモリや，ダイナミック・バス・サイジングなど機能強化された面が大きく，もし，ソフトウェア・モデルが違っていれば，全く別のプロセッサであると言いたいほどである。

勿論，併せてミクロン・ルールも下げられておりこの縮小化作業は現在も続いている。発表当時は，1.5μmルールで，サイズは約9×10mmであった68020も1.3μmルールへと変わろうとしている。

これ以外にもマスクの最適化の作業も継続して行なわれている。有名な話として，モトローラのセカンド・ソースであるトムソン・CSF（仏）〔現在のSGSトムソン〕は，政府のMILスペックで12.5MHz版の要求に対し，マスク・レイアウトの全面見直しをもって応え，チップ面積を3割減じて，16MHz版を作り，これで12.5MHzのMIL版を可能にしたとされている。これは，チップ面積の減少と高速度化は，LSIにおいて同じ意味を持つ好例である。

3-6 MPUの内部実行

68000ファミリではMPUの機能はいくつかのある程度独立した機能ブロックに分けて実行されている。まず，MPUのバスを用いて命令がデータをアクセスする機構，このアクセス時にアドレスを計算する機構，読込んだ命令を解釈する機構，これを行なう機構，割込みを扱う機構，等々がある。そしてこの分割は他にも例は多いが，特徴となるのはその重要部分がマイクロ・プログラミング技法で作られていることであろう。

これは丁度，68000というシステムがあり，データ・レジスタや，アドレス・レジスタという内部RAMと，ごく簡単な命令実行用MPUを想像していただくと比較的近いイメージとなる。このシステムは外部バス，つまりI／Oから自己のシステムに対する指令（命令）を受けて，これを内部の命令の起動に変えて動作する。指令により外部と更にやりとりも行なうし，RAM上の（データやアドレス・レジスタ）データの演算も行なう。これらの動作が外側からみると，MPUが，その機能を果たしている様にみえる，という事である。

68000や68020のチップの写真にnROMと記載のある部分が，この内部のマイクロ・プログラミング命令の格納されている部分である。ここは，他の一般のマイクロ・プログラミング例にもれず，ワード幅の広い，そして高速の（クロックの2倍）動作速度を持つロジックとなり，半クロックで，この内部の1動作を行なっていく（11ページ）。

68000ファミリで，比較的容易に上級バージョンが出たり，アルゴリズムが改善され得るのは，このnROMのプログラム変更と，それに伴う回路変更で済むからである。

第4章

一般的OSの使命と，その役割

4-1　一般的なＯＳ

コンピュータ・システムを簡便に使うためには，ユーザーやシステムの開発者は，それぞれの専門分野をシステムに乗せる，つまりシステムに専門作業を代行させる事になるが，専門家にとり，コンピュータ・システム自体に関する学習は少なければ，少ない程のぞましい。そこで，システムを使いやすくするためのものとして，システム側でシステムの専門家でない人々に近づき，かつ，システムの効率の良い運営を行なえる様に考えられているものが，ＯＳ＝オペレーティング・システムである。第４章ではこのＯＳを概説するが，個々のＯＳの詳細には立入らないので，ＯＳについて知識のある方々はこの章は不要であると思われる。

コンピュータ・システムでは，通常，何らかの形のＯＳが働いている。もちろんシステムによってははっきりしたＯＳの形をとらずに働いているものもあるが機能としてのＯＳに変わりはない。はっきりとした形を持たないものは，システム組込み用に使われるワンチップＣＰＵに多いが，このようなものでも，最近では整然としたＯＳの形態をとるものが増えている。

さて，16／32ビットのＣＰＵが働く環境となると当然，ＣＰＵの行なう仕事も複雑化し，プログラムも大規模になり，高度な作業をいくつにも分割されたソフトが，各々の役割りを同時に遂行していく形になる。

また，アプリケーションによっては同時に多数のユーザーの仕事を行なう事になる。この様なシステムでは，仕事の実行，中断や，どのユーザーにどれだけサービスするか，どれだけの資源をどの仕事に使わせるか等を決定し実行させるのが，ＯＳに与えられた使命となる。

コンピュータの利用分野が広がるにつれて，コンピュータの利用状況も変わってきている。コンピュータ利用の初期には，コンピュータ本体内には実際に使われるプログラムのみが存在し，コストのために，わずかな部分も切りつめて作られていた。しかし，コンピュータ自体が複雑化し，高性能化するに伴い，ユーザーやプログラマが，コンピュータのプログラムを全て管理する事は不可能であるし，非合理な事となってきた。またコンピュータのメーカー側でも，ユーザーのためのソフトウェアのサービスを提供する場合に個々のユーザーの要求に応えていたのでは非能率な上，必ずしもハードウェアを高度に利用するものとなり得ないものが多かった。そのため，このユーザーとメーカーの双方にメリットのある方式が作りあげられた。これが，現在一般に，ＯＳと呼ばれ，現在ではコンピュータのシステムにはなくてはならない最も基本的なソフトウェアの部分となっているものを採り入れたシステムの有り様である。このソフトウェアをコンピュータのメーカーは自社の製品系列に標準化して搭載する事により生産性を高め，自社のシステムの性能を充分に引出す基本ソフトをユーザーに提供する事によりシステムの高度利用を保障でき，個々のユーザーの要求にはＯＳ内の一部のテーブル，表，処理ルーチンの変更で対処し，基本部分を統一する事で速やかに安いコストでシステムの変更に対処できる様になった。もちろん，このＯＳを使用者の立場から，標準化して，仕様を決め，これをソフトウェア専門業者やメーカーに作らせ，メーカーの異なる機種にも同一のＯＳを搭載させる方式もある(UNIX－OS等)。この方式はユーザーがコンピュータの機種，メーカーを意識しなくて済むので，とても好まれている。しかし，各メーカーにとっては，多少の負担増になり(予め決まっている仕様を完全に満たすのは，かなり難しい）自社の機種のメリットをユーザーに良く知ってもらえないという欠点も持つ。ただしＯＳの選択もユーザーのコンピュータを選ぶ時の選択条件の一つなので，ほとんどの中型，大型機のメーカーは自社独自のＯＳと，世の中の主流を占めるＯＳとを用意するのが標準となっている。ユーザーは自己の応用の分野により適切と思われるＯＳを採用すれば良い。

4-1-1　ＯＳの効用

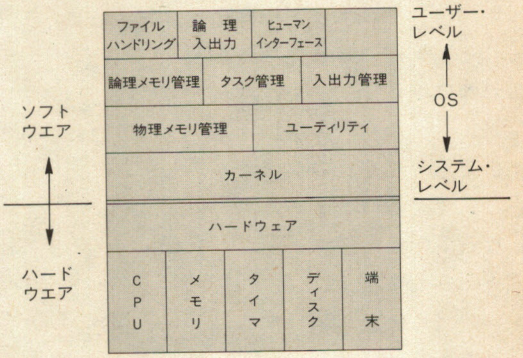
このＯＳとは，コンピュータの運営，管理をコン

ピュータ自身に行なわせるための部分であり，システム全体の効率的な運用を図り，使用者，ソフトウェアの開発者に快適な環境を提供するためのものと考えられている。さらに，このＯＳの基本的な考え方には，ユーザーや，システムの開発側と，ハード及びソフトのメーカー側との作業分担を明確にする事も一つの目標とされている。つまり，ハードウェアや，システムの標準の入出力はＯＳが管理するのでこの部分はシステム・メーカーの責任であり，ＯＳの仕様にあった応用ソフト，ユーザー側専用の入出力はユーザーの責任であると，分離する事が可能になり，各々のユーザーは標準的なシステム部分に関する詳細な知識なしに応用部分を作れる様になった。

つまり，ＯＳは，入出力装置，コンソールやプリンタ，電話回線の制御装置等の単純な装置や，外部記憶としての各種磁気ディスク装置や磁気テープ装置(フロッピー・ディスク，ハード・ディスク，MT)の制御と利用，コンピュータが働く時，プログラムを置いたり，データを置いたりするメモリの割当て，MMU（メモリ管理用ハードウェア）を持っているシステムでは，その管理，更にシステム時計の管理や，各ユーザーがどれだけ使ったかという記録をとる作業，一つの出力装置に複数の利用が集中した時の出力順の制御や，スプールと言って，後にまわした出力データをＯＳ側で保存する作業，個々のユーザープログラムへのサポートとしてファイルの管理，時間のサポート，割込みを個別の処理に接続するサポート，データを他のタスクへ転送するサポート等々，様々な作業をサポートしてくれる。

この様な作業を総括してシステムの資源の管理と言う(図4-1-1参照)。

図4-1-1 ハードウェアとソフトウェアの接続



4-1-2　マン・マシン・インターフェース

近年のＯＳはシステムでの作業を行なう人間へのサポートとして（人間も，システムの入出力の一部と考えられる。）マン・マシン・インターフェース部，つまり人間が関わる部分の作業，処理の改善が一つの重要な課題となっている。

このヒューマン・インターフェースともいわれている部分がシステムの成熟とともに増大，進化してきている。ユーザー側から見ると，システム構成の異なるシステムを操作する，または，その上のプログラムを実行する時に，もしＯＳが同じヒューマン・インターフェースを持っていればシステムの操作の習熟が不要であり，他のシステムでの経験，熟練が生かされる。

更に，ＯＳは，各高級言語（もちろん，高級言語が使える様な高度のシステムでは）のサポートも行なうので，他のＯＳで作成された高級言語で記述されたプログラムはソース・コードを移植すれば，変更なしか，比較的わずかな変更で新しいシステム上で動作する。これは各ＯＳが，それぞれ標準的な高級言語とそのライブラリを組合せて有しているのでこの組合せにより，高級言語の標準関数とＯＳ内のサポート・ユーティリティの相関がとられており，ソース・コードを再度コンパイル・リンクする事によりそのＯＳ下で動作するオブジェクトとする事ができるからである。これはユーザーにとって大きなメリットがある。

4-1-3　種々のＯＳ

コンピュータの応用分野の広がりに応じて，ＯＳにもいくつかの使用目的を持つものが作られる様になった。

その代表例として，秀でたマン・マシン・インターフェースを持つUNIX・OSがある。この他にもプラント等の製造ラインなどで使われるプロセス・コ

ントロール用のＯＳ，オフィスや事務所で使われるＯA用のＯＳ，ロボットや自動機器に組込まれて用いられる制御用のＯＳ等がある。

この他に現在では，小はポータブルのパーソナルコンピュータから，大は銀行等で用いている超大型機にまで，それぞれのシステムの大きさ，使用環境に応じた専用のＯＳがある。

８ビットの小型のマイクロコンピュータの中で，開発ツール等に使われたＣＰ／Ｍ，現在パーソナル・コンピュータに広く使われているMS－DOS等が有名である。

これらの小型のＯＳはそのほとんどがUNIXの縮小版であり，１人のユーザーをサポートするべく作られたシングル・ユーザーＯＳと言われるものである。

さらに，現在ワープロとか，ＴＶゲーム機といわれているマイクロコンピュータの製品があるが，これらのほとんどに何らかのＯＳが使われている。

新しいタイプのＯＳとしては，映像化の進歩により，エンジニアリング・ワークステーション（ＥＷＳ）やコンピュータ・エイデッド・デザイン（ＣＡＤ）で使われる画像処理機能を持つグラフィック・ＯＳ（AEGIS:アポロ・コンピュータ）等がある。

4-1-4　ＣＰＵの高度利用

ＯＳは更にユーザーのＣＰＵの使用効率を高める為の作業も行なう。その代表的なものは，プリンタの為のスプーラと，バッチ処理である。この双方とも人間が操作しているコンピュータの表面的な処理とは別に，ＯＳのカーネルの様に隠れて働き，要求された仕事を行なう。まず，スプーラを利用する事により大量のプリンタへのリスト出力を行なう時，ＯＳに対して指示を与える。ＯＳは，そのプリンタが使用中であるとなしとに関らず，ＯＳ内の作業を行なって，データをスプール内に入れて，制御を人間に戻す。この作業は，ほぼ一瞬で終了し，人間は次の作業に進み，時間のかかる作業であって，コンピュータの能力もそれ程必要としない作業（プリント・アウト）を並列に行なう事ができる。バッチの方は，もう少々，作業者の意図が反映される。作業者は，型式化した作業で，余り遂時の指示のいらない作業の手順をリストにしたものや，ある一つの命令をバッチに投入する（これは，VERSA-DOSであれば，命令の先頭に「@」を付ければ良い）するとこのバッチ・プログラムは，コンソール・ターミナルから引き続いて入ってくる作業と作業の間の空き時間，つまりＣＰＵの無効時間があると，この時間をこのバッチに投入されている仕事に割り振って，その仕事を行なわせる。この方式は，例えばコンパイル―アセンブル―リンクといったプログラムを処理する時の様な場合に向いている。これらの作業では，ファイル名以外には外部から与えるパラメータが少ないからである。この種の作業は多く，統計の計算や，自動配線のプログラム等々がある。そしてこのバッチの利用で，ユーザーはコンソール・ターミナルから，遂時指示を行なう必要のある作業，例えばエディタによる編集，作成，等々の作業を並行して行なう事ができる。人間は，その作業に対してのコンピュータの応答を最速で要求するが，それを実現していると実際のコンピュータの能力の利用率が数％にすぎず，その他の時間は単に指示待ちとなってしまう。コンピュータ・パワーという資源の浪費を減らしてシステムの高度利用を図るのもＯＳの作業の一つである。

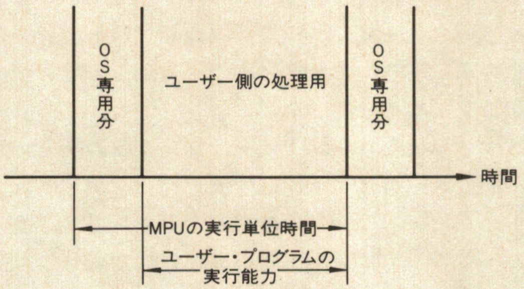
4-2　ＯＳの仕事

以下にＯＳに課せられた仕事の内容を，分類し概説する。

4-2-1　仕事（TASK）の実行，中断の決定，管理

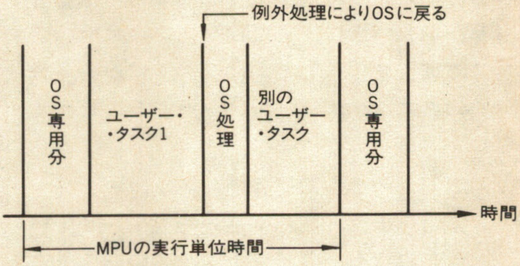
一般に各タスクは，実行の要求によりスタートする時は，ＯＳから実行用の時間をもらうという形になる。この実行時間の割当ては，そのタスクに与えられている優先順位と，状況（アクティブ，休止，入出力の対応等）等により決定され，実行時間を持ったタスクは，ある所定の時間または自身の連続的な実行ができない状況になるまで実行する（実行できない状況とは，一般にスーパーバイザ・コール＝ＯＳに対するサービスの要求という形をとり，ＯＳ

図4-2-1　実行時間の割当て



MPUは単位時間内に平均して実行できる命令数はほぼ決まっている。そこで実行用の時間＝実行できる命令数＝処理能力となる。一定時間ごとに割り込みを入れて，OSに処理を移す様なシステムでは，ここで処理能力をOSとユーザーに分ける

図4-2-2　TRAPによりOSにもどる



あるユーザー・タスク1を実行中に，TRAP命令等の例外処理によりOSに戻り，このユーザー・タスク1を中断して，別のユーザー・タスクを実行する

内にプログラムの実行は戻ってくる)。そこで，制御をＯＳに渡すか，何らかのＯＳのサービス待ちになるのが通例である。ＯＳは，別のタスクに実行時間を与える，あるいは割込み処理をして，条件が整えばまた元のタスクを継続的に実行させる（図4-2-1，2参照）。

4-2-2　システムの資源管理

ＯＳに関して資源という言葉は，システムのハードウェアとソフトウェアの総体から，ＯＳの専有部分を差し引いたものであり，物理的なＲＡＭの量やドライバ・プログラムを含む入出力（論理入出力)，ディスクによるファイル・システム，時間管理のシステム，プログラムを実行できる能力，等々……のユーザー・プログラムに与えられるサポート・サービスの総体をさす。

ターミナルに向けて，キーボードより，「エディタ」を意味する文字列を入力すれば，各キー入力は，ターミナルのサポート・ハードウェアから１文字ごとの割込み要求となり，ＯＳは，この割込みをサポートして，ＯＳ内部の文字列バッファに入力された文字を並べてゆく。そして，入力された文字が予め決められていた１行の終わりを示す文字であった時に事前にあるユーザーからの，その文字列の要求がなければ，文字列の内容を解析するプログラム部分（ＯＳの機能）に引き渡す。このプログラムは，ＯＳに対する命令（コマンド）として文字列を扱い，対応する文字列のコマンドがあれば，これを，やはりＯＳ内のファイル・システムを用いて呼び出し，メモリ上に置いて実行可能な状況とする。ここではエディタ・プログラムがディスクより読み出されメモリ上に置かれて，実行がかけられる。そして，その時，文字列の残り部分が，このエディタに引き渡され，エディタ自体による解析が行なわれる。この文字列の残り部分で，何らかのソース・ファイルを自分が使えるメモリに読出してくる事をＯＳに要求する。またターミナルにスタートのプロンプトを出力し，読んできたソースの文字列を表示する場合に，エディタ・プログラムはＯＳに表示すべき文字列を引き渡す。エディタによっては，何分間かキーボードからの入力がなかった時に，自動的に自分自身の実行を打ち切るものがあるが，この場合にはＯＳが管理するシステム時計によって与えられる時間情報が用いられるし，自己打切りの作業もディスクへの書込み，メモリの解放，等々，ほとんどＯＳの作業である。

これらの作業を通じて，ユーザーはシステムの資源を利用している訳であるが，他のユーザーとの競合や，ディスク，入出力の重複利用，等々，一元的に管理を行なう必要性のある所は多い。この様な管理をシステム資源の管理と言い，限られた能力を有効に，かつ易しく利用するための環境整備である。

4-2-3　エラーの管理

ユーザー・タスクの実行中に発生したエラーに関

し，リカバリの処置またはタスクの中断といったエラー管理を行ない，同時に多数存在している他のタスクへ影響が出ない様に，また親タスク（エラーの出たタスクの実行を要求した別のタスク）がある場合には，そのタスクへエラー・メッセージの転送等を行なう。

高級なＭＰＵシステムでは各種のエラー・チェックがＭＰＵに組込まれ，内部割込みを発生させているので，これらのサポート・プログラムの形をとる事が多い。

4-2-4　ＯＳ資源に関するサービスの提供

ＯＳは，基本的には各種の物理的（ハードウェア上の）資源を独占しているため，ユーザー・タスクは自身の実行にハードウェアを必要とする場合に，自分自身で直接ハードウェアを操作する事はできないので（高級なマシンでは，ユーザーの実行環境上には物理的資源はメモリしかない)，ＯＳを経由してハードウェアのサービスを受けることになる。

この理由は標準的な入出力に関しては個々のユーザーがハードウェア，それも各種の異なった内容のものを個々に製作する不合理を避け，システムの規模の変更やメモリの変更に対応して，ユーザーが細かい修正を行なわなくてもよい様にするためと，ユーザーがハードウェアのドライバ・ソフトウェアについて，重い負担をしなくても済ませられる様にするためである。

したがって，ＯＳには各種の物理的入出力のドライバ･プログラム，メモリ管理ユニットのドライバ･プログラム，DISKがある場合にはDISKのコントロール・ドライブ・プログラム等々が組込まれ，各種サービスをユーザー・プログラムに提供する。なぜこのような構成をとるかと言うと，ある一つのマシンに一つのＯＳが乗っており，一つの固定した動作環境が提供されても(従来のメーカー製のミニコン，現在のパソコン等)，ユーザー･プログラムによっては，それぞれの入出力の形態，利用度，入出力装置の専有度は大幅に変わる。したがって，特に複数のプログラムで利用されそうなものは，ＯＳが管理して重複使用をさけ，優先順位をつけてユーザー・タスクに利用させ，または利用を中断させて，他の利用にあてる等の管理を行なう。

主記憶中のメモリについても同様で，ＯＳは各タスクの要求に応じ，またタスクの優先順位を計算して主記憶を割当てる。

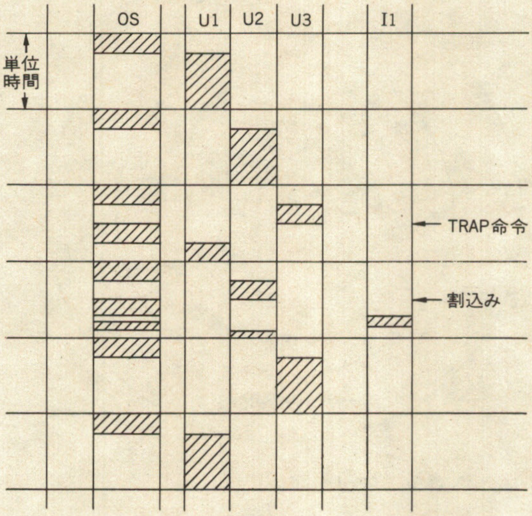
こうする事により，ユーザーはプログラム開発時に標準的な入出力については作成デバッグの必要はない。またハードウェアの状態が変更されても，違ったハードウェアに対応するＯＳの責任であり，ユーザー側は何の変更もいらない。更にデバック時にダミー（仮想上の入出力の設定）の処置が楽である等，長所が多い。全体的に見ても，ＯＳから各種のサービスが受けられ，これらのサービスはシステムが変更されてもＯＳが同一であれば基本的な部分の変更はないため移植性が高く，ユーザーのシステムの変更や更新により良く対処できる。

4-2-5　割込みの処理

これは，いくつかのレベルに分けられるが，外部からの割込みはハードウェアによるため，4-2-2の資源の管理と深く関わりＯＳの仕事となる。ハードウェア割込みは，この他にも4-2-1のタスクの実行／終了のタイミングを決めるための，一定時間ごとのリアルタイム割込みもあり，ＯＳの動作をサポートしている。

内部からの割込みは，８ビット程度のＭＰＵでは数個のソフトウェア割込みに限られていたが高級なＣＰＵでは一般にこの能力は大きく拡大強化されている。そして，内部発生の割込みにも各種のレベルができてきて，実行上のエラーを示すもの，ソフトウェア上の条件により発生するもの，単純なソフト割込みという様に大別する事も可能となる。ソフトウェア割込みは通常，ＯＳに何らかのサービスを要求することであり，サービス内容を示すデータ・テーブルをオペランドとして伴って要求され，ＯＳは要求内容をチェックして実行する。もちろん，要求元のタスクは要求したサービスが満たされるまで待期に入る場合と要求のサポートは表にでずに実行され，タスク自体は継続して実行を先に進める場合とがある（図4-2-3参照)。

図4-2-3　マルチタスクを実行している様子



単位時間の替り目，例外処理の起動で必ずOSに入り，
そこから各タスクへ分岐する
ここでU1,U2,U3はユーザー・タスク，I1は割込処理を表す
（割込処理はユーザーが特定の割込みを扱う事を宣言
した場合の例で，OS内で処理されるものもある）

エラー処理の内部発生割込みも幾つかのレベル，種類がある。重要度によりレベル分けされ，システムの実行に支障を起こす重大なものを最も重要なものとして扱い，次に何らかのソフト上，実行上のエラーと思われるもの，データの異常によると思われるもの，ソフト上のチェックによるもの，通常ソフトウェア割込みとレベルは下がってくる。この細かい点については，第10章の「割込み及び例外処置」の項を参照されたい。

4-3　ＯＳの実際

ＯＳは各メーカー，ソフトウェア会社から種々のものが発表，使用されているが，基本は旧来ＩＢＭ社の発表した形を保っている。この古典的なＯＳの形態を保っているＯＳは多く，68000用のＯＳとして，モトローラ社から発表されているVERSA－DOSもその一つである。このＯＳにそって，基本的な形のＯＳの概略を説明する。

4-3-1　カーネル

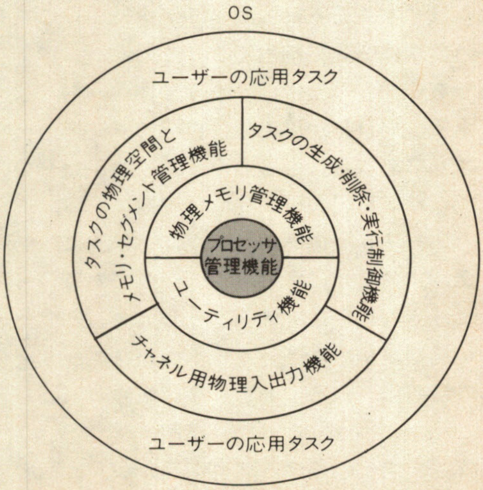
まずＯＳではプログラムがコンピュータ・システムの上で動作し何らかの仕事をする時，この仕事をタスク（TASK）とかジョブ（ＪＯＢ）と呼んで仕事の一つの単位としている。

このタスクの管理，つまり，タスクの起動，実行，中断，待期，消滅等のプログラムの実行状況を管理する部分がある。この部分は，ＯＳ内の種々のタスクの制御も行なうので，システム・ソフトの最も基本的な部分となる。これを「カーネル（KERNEL）」と呼んでいる（図4-3-1参照）。

システムの構成が異なっても，一般にカーネル本体は変わらない。ただし，このカーネル・ソフトが働いている間は，システムは自己が何をするべきかを決めようとしている時間なので，全体から見ればまったくのデッド・タイム，つまり，何もしていない事になる。従って極力この時間を短縮すべく多大な努力が払われている。そこでよく，システムの性能評価にこのカーネルのタスク切替えに要する時間が出てくる訳である（図4-2-2）。

このカーネルはこれ自体入出力も持たず，ハードウェア，ソフトウェアの割込みによって起動され，各種の処理を行なう。これは，システムの入出力や各資源（システムの利用可能な部分）は，イニシャ

図4-3-1　OSの中心，カーネル部



ライズ後は，割込みにより動作する事と，各種のユーザー・ソフトウェアからのサービス要求はソフトウェア割込みを使って行なわれる事によっている。これらの割込みを受けて，それぞれの要求内容により，次のレベルのサービスに引渡したり，ハードウェアのサポート・ルーチン・プログラムへ引き渡したり，自分の内で，タスクを停止したり，別のタスクをスタートさせたり，時間待ちをさせたりする。

このうち，タスクを切替えたり，スタート／ストップさせる機能をタスクのディスパッチ機能と言いこれを行なっている区間をディスパッチ・サイクルと言う。このカーネルは，システムが立ち上げられる時，一番先にロード／起動され，これに各種のシステム資源を接続してゆく事からイニシャライズは始まる。そして，各種の割込みへのサポートを開始する。

このサポートはＯＳの内部からも，ユーザー・プログラムからも利用され，タスクの流れの変更については，ここが一元的に管理する。

一般的に，カーネルは，その内にタスク管理テーブルを持ち，これによって実行の優先度，実行状況等々を参照して，そのタスクを実行させるかを決めてゆく。

4-3-2　メモリ管理機構

カーネルの次にくるのが，メモリ管理機構である。システムは必ず，あるＲＡＭエリアを持つ。これをその時の要求に応じて割当てていくのが，この部分の役割である。もちろん，メモリ全体のうち，まず第一にカーネル自身が優先で，自己の駐在場所とワーク・エリアを取る。もちろん，カーネルがＲＯＭ化されていれば自己の駐在場所は不要である。次には，このメモリ管理機構が，また，その次には……という様に，メモリは常時駐在の必要のあるＯＳの部分によって占められていく。そして，ユーザー・タスクが起動がかかると，カーネルが各ユーザー・タスクの初期化，スタートを行ない始める（図4-3-3）。

この時，まず起動されるプログラムがＲＡＭの上で動くプログラムなら，その駐在エリアを割当て，

図4-3-2　システム効率の為のメモリ管理部

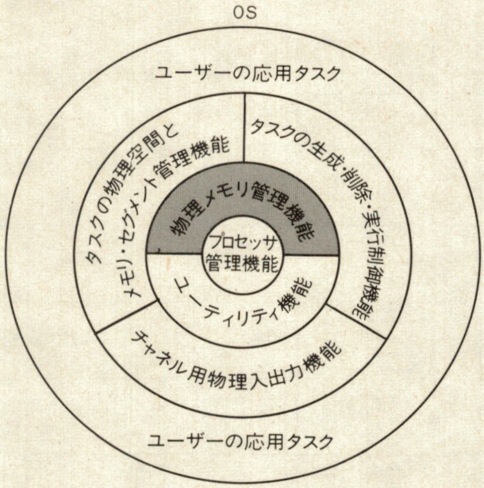
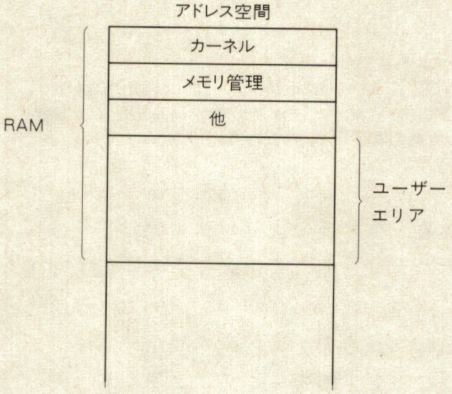


図4-3-3　OSのメモリ空間使用例



要求されたワーク・エリアを与える。このユーザー・プログラムが実行されて，更にソフトウェア割込みにより，追加ワーク・エリアの要求があればこれを与え，タスクが終了すれば不要エリヤとしてこのワーク・エリヤやタスク・エリヤを取上げ，他へまわす仕事をする。

もちろん，その内には，与えるべきＲＡＭがなくなる事もあり，その場合にはエラーを要求側へ戻すか，他のタスクが終了して空きができるまで待たせておく。このように，システム中にあるＲＡＭという資源の管理を行ない，システムの効率の良い動作を行なわせる部分である（図4-3-2）。

図4-3-4　システムへのサービス/ユーティリティ部

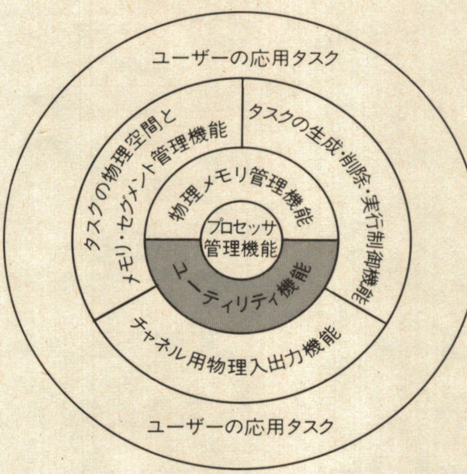
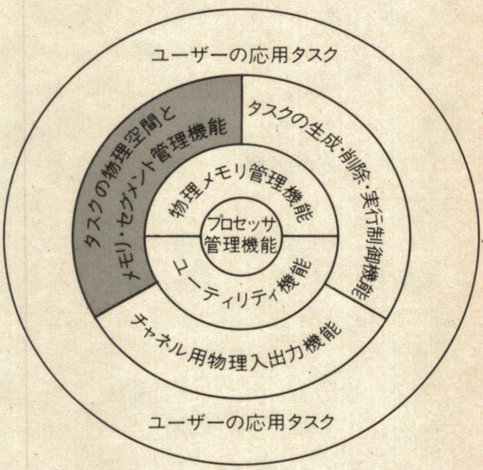


図4-3-5　MMUを有する場合のコントロール部



4-3-3　ユーティリティ機能

この部分はＯＳの行なう基本的なサービス・プログラムである。ＯＳは，タスクの制御に伴って，タスク間通信，タスク間でのデータの転送，タスク番号，時刻，日付の管理等，様々なサービスを提供する。そのうち，このカーネル部がサポートすべき基本的なユーティリティがここに含まれている(図4-3-4)。

4-3-4　アドレス空間とメモリ・セグメント管理

ここは，タスク自体が持つ固有のアドレス空間とメモリ空間をセグメントに分割して制御するための部分である(図3-4-4)。

各タスクは，プログラム・コードの部分とデータ・エリアの部分，及び，タスク制御のための非同期処理キューの部分があり，この各々が，メモリ・セグメントとして管理される。このメモリ・セグメントは，「書込み禁止」とか，「読出しのみ」などの属性を持ち他のタスクと共有部分を持つこともできる。もちろん，これらは論理上のアドレスも持っている。この諸条件を満足する環境を作り出すのが，この部分に与えられている使命である。

従って，各タスクの切替えや生成，消滅に伴い，また，各タスクからの要求により，レベル１の物理メモリ管理を利用して，論理—物理アドレスの変換を行なわせ，各タスクごとのアドレス空間を管理する。もちろんこの作業に伴い，MMUがあれば，その管理も行なう(図4-3-5)。

4-3-5　タスクの生成，削除，および実行制御

通常，システムのイニシャライズ時に，いくつかのサーバ・タスクとユーザー・タスクがイニシャライズ・タスクによりスタートさせられる。これらのタスクにより，システムの基本的な動作が始まり，その後はユーザーの要求，事象の発生等によって，次々と新しいタスクを起動し，仕事が終われば消滅させてゆくが，これらのタスクの生成，起動，状態待ち，時間待ち，保留，停止，消滅を実行するのがこの部分であり，ソフトウェア割込みにより，これらの作業が起動され実行される。

この部分は，カーネルのディスパッチの細かい実行をサポートする延長部分である。

4-3-6　物理入出力

ここで，物理と言っているのは，ＯＳ内ではタスク間においてデータ，フラグ，キューのやりとりがあり，更にファイル・システムまで含めた論理入出力もあるので，各タスクから見ると入出力は必ずしも実際の装置，周辺ディスクによらないからである。ここでは各種の周辺入出力回路のハードを制御する

ソフトウェア・ドライバである。この部分は同じOSであっても最も変更の可能性の高い部分である。それは，カーネルはシステムによらないし，メモリの量は，システムが自己で計測する場合が多いのでこれもシステムによらない。そこで，前述の部分でシステムにより変更が必要になるのは，最大タスク数とか，リアルタイム・クロック，MMUの有無位の所である。ところが，ここで扱う入出力はシステムによって最も大きく変わる部分だからである。

ここは，具体的には各入出力ディバイスごとにディバイスのイニシャライズ・プログラムとドライブ・プログラムを作り，入出力の起動としてイニシャライズ・プログラムを1タスクとして，生成一起動一停止一消滅させ，ディバイスの初期化を行ない，割込みを利用する場合には，カーネルに対し，ディバイスの出す割込みを処理するプログラムを接続する。この割込み処理タスクがディバイスとの実際の入出力を行なう(**図4-3-6**)。

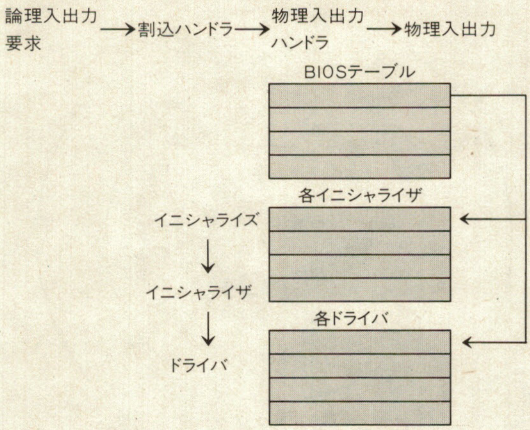
この際，もし入力なら，得たデータをOSの通信機能を用いて，データ処理用のタスクに渡す。出力なら，データは通信機能を通じて受取り，このデータでディバイスをドライブする。割込みを使わない場合は，速度，効率は落ちるが，一定時間ごとに起動されるサブ・タスクを用いて，ディバイスを監視し，サポートを行なう。後は割込みの場合と同一である。

基本的な入出力ドライブは，上記の様に働き，各種の入出力，ターミナル，プリンタ，ハード・ディスク，フロッピー・ディスク，マグネット・テープ等々を利用する。

この様な実際に基本的な部分のプログラムを「BIOS」Basic Input Output Systemと呼んでいる。システムを新たに立ち上げる時，または特殊な入出力をOSにサポートさせる時は，各OSの規定に乗っとったイニシャライズ・プログラムとドライブ・プログラムをこのBIOSに組込めば良い。

この組込み方法には2通りあってOS自体を全部組直してしまう方法と(例：VERSA−DOS)さらにOSに追加できる様になっている方法とがある(OS9)。

図4-3-6　OSから，各入出力の起動使用手順



全部組み直す事をシステム・ジェネレーション(SYSGEN)と呼ぶ。

追加できる方法は特定のIO名ファイルがあってこれを変更，追加する事によるのが普通である。

4-3-7　その他のサポート

OSは，これまで説明したカーネルを核とした部分が必要最小限の部分であるが，UNIX−OSの様に開発用のOSや，OA用のOSでは，ここにシステムを使いやすくするための論理入出力や，ファイル管理，人間の操作に合せたインターフェース等が加わって，OS全体となる。これとは逆にFA用のロボットや，自動計測機に使われるOSは，人間が操作をせず，機器間での通信や表示が主な入出力となるので，人間用のインターフェースは重視されない。

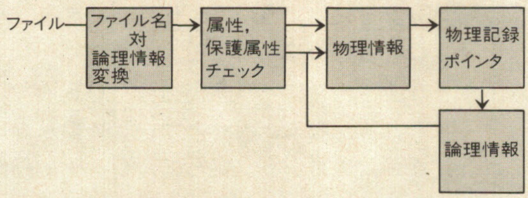
もう一つの重要なサポートはソフトウェアのデバッグである。OSはエラーの管理も行なっているのでこの機能を拡大してのソフトウェアの実行モニタとしても使えるのでデバッグのサポートも大切な仕事である。

4-3-8　ファイル管理

ディスクを持っているシステムで，ディスク上のデータ管理を行なっているタイプでは，ほとんどのシステムが何らかのファイル管理を導入している。

ファイルとは，OS上で管理されるデータ，プログラム等を，ひとまとめにして，単位ブロックとし

図4-3-7　OSの論理ファイル検索手順



たもので，ＯＳに対する各種のサービス要求はこのファイル単位でサポートされる。

このファイルは，文字で構成されているもの，機械で扱うコードで構成されているもの，プログラム・コードより成るもの等多くの種類がある。ＯＳはファイルごとに消去禁止や，書込み禁止，特定ユーザーのみに使用許可等々の各種保護を行なう（図4-3-7)。

この利点は，ファイルの内容はディスクの上では一つのファイルであっても一般に複数個所に記録されている。もし直接これを読出そうとすると，ポインタで接続された連鎖を追跡しなければならない。ところが，ＯＳのファイル管理（ファイル・ハンドリング・システム＝ＦＨＳ）の助けを借りれば，単にファイルの名前を指定するのみでそのファイルを指定でき，ユーザーは物理的な情報には関与しなくてすむ。実際にユーザーがシステム上で種々の作業を行なうとファイルの数は相当大きくなるので，ユーザーが物理的記号を管理するのは不可能であると思われる。それが，各ファイルが複数のエリアに分かれているという実際のディスク上の内容では，ますます不可能である。

4-3-9　ディレクトリ

これらのファイルも数が増えてくると，各ユーザーは，ユーザーごと，仕事ごとにファイルをグループ化して管理したくなる。そこで，この管理を行なわせるために，グループ化したファイル群を，垂直，水平に並べられる様にしたものが「ディレクトリ」である。

「ディレクトリ」とは目次のことであるが，一般文書の目次が，章や項の名前とページ数の対応表になっている様に，ディレクトリは，ファイルまたは次の層のディレクトリの名前とそれらのディスク上の位置情報から成る（図4-3-8)。

これにより，ディスクのハンドラ・プログラムは論理的位置情報を得て，これを，ディスクのシリンダ番号，トラック番号，セクタ番号の様な実際の位置情報に変換して扱う。

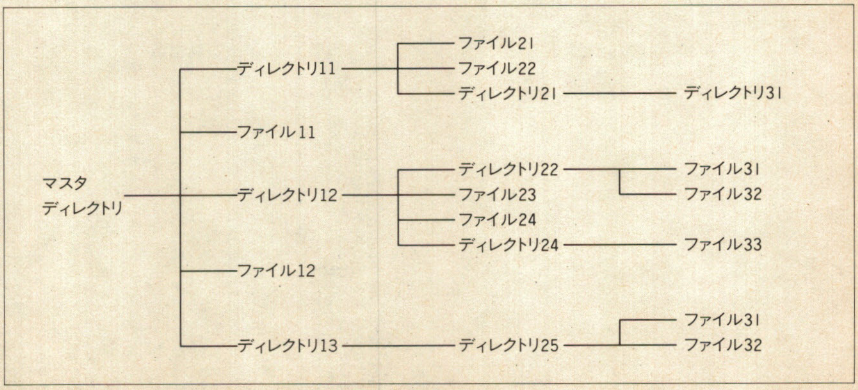
ディレクトリが次のディレクトリを指している場合には，そこから更に次の階層を捜して対象のファイルを検索してゆく。これにより，ファイルの論理的な分類と位置情報を形作る事ができる。

4-3-10　エラー管理

これ以外にも，高級なプロセッサは，エラーを検出するとソフトウェア割込みを起こす。また先のテーブル等にも，作成上，記載上のエラーは，しばしば発見されエラーとなる。

この様なシステム上のエラーに対処し，エラーの処置を行ない，エラー情報を送り返すのもＯＳの重

図4-3-8　ディレクトリとファイルの関係



要な仕事の一つである。特に応用ソフトウェアの作成時，ＯＳ内で行なわれる作業については，デバッグができないので，このエラー情報が最も有力な手がかりになる。また，ユーザー側は，返ってきたエラー情報により，自己のタスクを止めるか，修正を行なうか，などの判断を行なう事ができる。マルチタスク・マルチ・ユーザーのシステムでは，このエラー管理によって他のユーザーの資源を保護しているので大きなシステムでは特に重要な部分である。

4-4　ＯＳへの仕事の依頼

一般にＯＳに何らかのサービスを要求する事を「スーパーバイザ・コール」とか「ＯＳコール」と呼び，ソフトウェアの割込みを用いて行なう。ＣＰＵが複数のソフトウェア割込みをサポートしている時は，その複数のレベルをＯＳ内の階層に対応させる事が良く行なわれる。更に，ソフトウェア割込み発生に際し，オペランドとしてデータ・テーブルを作成して，これに各種パラメータを記載して，サービス側へのデータとする。このテーブルを作成する作業がＯＳコール＝スーパーバイザ・コールを使うための主な仕事となり，マクロ・アセンブラや，高級言語のライブラリは，ソフトウェア割込みに伴うテーブル作成のプログラムが大半を占める事になる。

この作業を通じて各ユーザー・プログラムは，別のプログラムを起動したり，外部と通信を行なったり，不要になると消滅させたりもできるし，ディスク上からデータを読んできたり，作成したデータをディスク上に書き込む事もできる。もちろん，シリアル・ターミナルの様な物理入出力や，ＯＳ内では更によく使われる論理入出力へのサービス要求もこれらの作業を通じて行なう。従って，ＯＳ下で動作するプログラムの自己の初期化と入出力は，ほとんど全てが，このスーパーバイザ・コールで占められこれを通じてシステムの各資源を入手，利用する事ができる。

逆に言えば，自己のプログラム内部の論理以外の全ての設定や，環境は全てこのＯＳのサービスによるもので，これらを受けられるのが，ＯＳの最大の効用であるという事になる。

4-5　マルチＯＳ

大型のシステムになると，多くのユーザーがシステムを使用する。この時一つのシステムに一つのＯＳしか乗っていない場合には，ユーザーはそのＯＳの操作を習わなければならない。

この不便さを解消するのが，マルチＯＳで，システムは基本的には一つのＯＳで動作するが，ヒューマン・インターフェースを中心にして，操作上，複数のＯＳが動作している様に見えるシステムである。この詳細は第５章マルチＯＳを参照いただきたい。

第5章

68000ファミリにおけるOSのサポート

5-1 スーパーバイザとＯＳ

68000ファミリ，特に68020ＭＰＵを使ったシステムは，何らかのＯＳを実装していると考えられる。

逆に言えば，このＯＳや高級言語がどれだけ作りやすく，また既存のコンピュータ・システムのデザインの上に容易に乗せられるかが，応用の重要なポイントとなってくる。

そこで68000ファミリでは，どの様なＯＳのサポートが考えられているかについて述べる。

68000ファミリは，始めからＯＳマシンを作る事が想定されているため，従来の下位プロセッサにはない特権命令(プリビレッジ命令)，各種内部自動チェックによる割込み等がある。その理念を知らないと取組みづらくなるので，特にこの章を設けた。

68000ファミリは，ステータス・レジスタ（ＳＲ）の中にＳフラグを持っている。このフラグに１が立っている時がスーパーバイザ・モードで，０の時はユーザー・モードと名付けられている。

二つの相違点は以下の点である。

1)　スタック・ポインタ（Ａ７）がスーパーバイザとユーザーと各々専用のものが使われる。

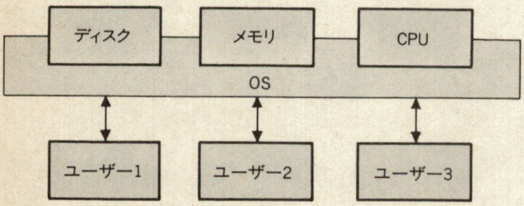
2)　命令の内，特権命令（プリビレッジ・インストラクション）と呼ばれるものが，ユーザー・モードでは実行できず例外処理へ移行する。スーパーバイザ・モードでは実行できる。

3)　ファンクション・コードの内，ＦＣ２が切替る。スーパーバイザ時１，ユーザー時０（これにより，外部で内部の状態を知る事ができる)。

4)　スーパーバイザ・モードではユーザー側のスタック（ＵＳＰ）を操作できる。

5)　スーパーバイザ・モードではオルタネイト・ファンクション・コード機能を使い，ユーザー側空間をアクセスできる(68010以上)。

図5-1-1　OSの資源管理



ユーザー・プログラムは，ＯＳにより，ハードウエアから切り離される。ＯＳは各種の資源を一括管理し，ユーザーに配分する

5-2 スーパーバイザへの移行

次に，どんな時にスーパーバイザ・モードになるか，その移行例をあげる。

1)　リセット時

2)　バス・エラー時

3)　内部発生エラー時

4)　外部割込み時

5)　TRAP等，ソフトウェア割込み時

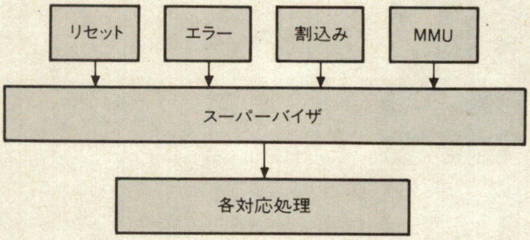
基本的には，68000ファミリの例外処理ベクタを使う場合には，スーパーバイザ・モードとなる。これは，その前のモードがユーザーであろうとスーパーバイザであろうと同じである(**図5-2-1**参照)。

以上の点から，汎用のＯＳの備えているべき機能について考えてみる。

1)　リセット時

ＯＳは一般に環境の整備をしなければならない。各種，入出力装置に対しての初期設定，ディスクの立上げ，メモリの初期化，スタックの設定等を行ない，ディスクや他の外部記憶にＯＳ本体がある場合には，ＯＳ本体のブート・アップ（メモリへの読込み）をも行なう。システムによっては日付，時刻の入力，システムの管理者による正規の立上げの確認のため，合言葉（パス・ワード)，暗号（セキュリティ・コード）の入力を要求する場合もある。

図5-2-1　スーパーバイザへの移行理由



外部ハードウエアからの様々な信号は，ベクタ・エリヤを通ってスーパーバイザ・プログラムの各々の対応処理によって処理される

また，大型ディスクを備えた装置の中には，ディスク内のワーク・エリヤの確保，クリヤ，不要のファイルの整理等を行なうものもある。

この様に，リセット時はシステム全体の立上げとイニシャライズとなっており，システムの本来の機能としてのプログラム（ユーティリティ）は一つも動作していない。この時，システムはOSモード，つまりスーパーバイザ・モードであると言える。

またこの時，MPUは割込みマスクのコントロール，ユーザー・スタックの設定のほか，各種の特権命令を実行する必要がある。

2）　バス・エラー時

バス・エラーは，外部の回路で各メモリ・アクセス・サイクルの時間幅を測定し，上限を越えた時にMPUにメモリのアクセスに対し，失敗が起こった事を知らせるハードウェア上の信号である。

メモリのアクセスが失敗する原因は，

a)　プログラムのミスで異常な所をアクセスし，そこにメモリがないか，MMUによって保護されていて正しいアクノレッジが得られない時

b)　プログラムのアクセスは正常だが，仮想マシンの構成上，アクセスした部分は主記憶上になく（つまりMPUのアドレス空間上のメモリ中にない），外部記憶（通常はディスク中）に存在している時，この時も，当然，メモリのない所をアクセスしているのだから，通常のアクノレッジは返ってこない。

バス・エラーの処理は，その発生原因が，a)のプログラム・ミスと判断した時（つまりb以外）には，そのプログラムの実行を中止し，エラーをユーザーまたは他のプログラムによって起動させていれば，そのプログラムに知らせて終了する。

b)の場合には，MPUにはアクセス・アドレスとどこのアドレスが外部記憶の中に入っているかの情報があるので，この二者をつき合わせた結果，外部記憶内の部分へアクセスすることが正しいと判断した時には，外部記憶から相当部分を主記憶メモリに移し，再度アクセスを行なわせる。

この作業も，システム・スタック内へのアクセス，タスクの打切り，システム・エラー情報の発生，「RTE」などの特権命令の使用，外部記憶の使用と，OS機能の主要部分となる。

3）　内部発生のエラー時

内部発生のエラーは，種類によってユーザー・プログラム内のエラー処理に引き渡す場合と，回復不能としてそのタスクを打ち切る場合がある。

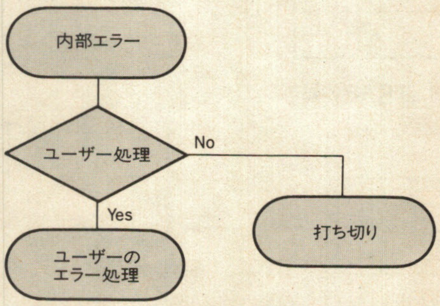
例えば，特権命令違反，不法命令コード等は回復不能だが，割り算の分母がゼロであった場合は，データ・エラーとしてユーザーのエラー処理に引き渡してやる事ができる。OSのサービスの一種にこのエラー処理を登録し，エラー処理をユーザー側で行なう事ができる様にする機能がある。

そこで，このユーザー側で処理を行なうとOSに向けて宣言されていないエラーは全てOSが扱い，エラーの種類，処理登録の有無のチェック，場合によりタスクの打切り等を行なうために，スーパーバイザ・モードを使う（図5-2-2）。

4）　外部割込み時

ハードからの割込みの許可／不許可は，割込みのマスク・レベル，回路に対する初期設定等で行なう。そこで，各割込みに対する処理の第一次はOSが行ない，OS内で処理されるもの，ユーザー側に引き渡されるものに区別され，また不当な割込みに対しての無視，またはエラー情報の発生等を行なう。この場合も，各タスクからの外部割込みの発生に対し処理の登録のできるOSもある。

図5-2-2　内部エラーの処理



MPU内部のエラーがユーザー・プログラムで起ると，ユーザー・プログラムがそのエラー処理を実行する事が宣言されていればその処理を実行させ，それでなければユーザー・プログラムを打切る

ＯＳの仕事の一部として，割込みを使って，入出力装置との間でデータ転送を行なう際に，ユーザー側の処理タスクと，データ・ブロックをやりとりする方法もよく使われる。これは，ユーザー側から見れば，ある情報ブロックの入力をＯＳに依頼する，または情報ブロックの出力をＯＳにまかせるという形をとり，ＯＳはその様なサービス要求を割込み処理タスクの起動という形で処理するというものである。

図5-2-3　OSの対応

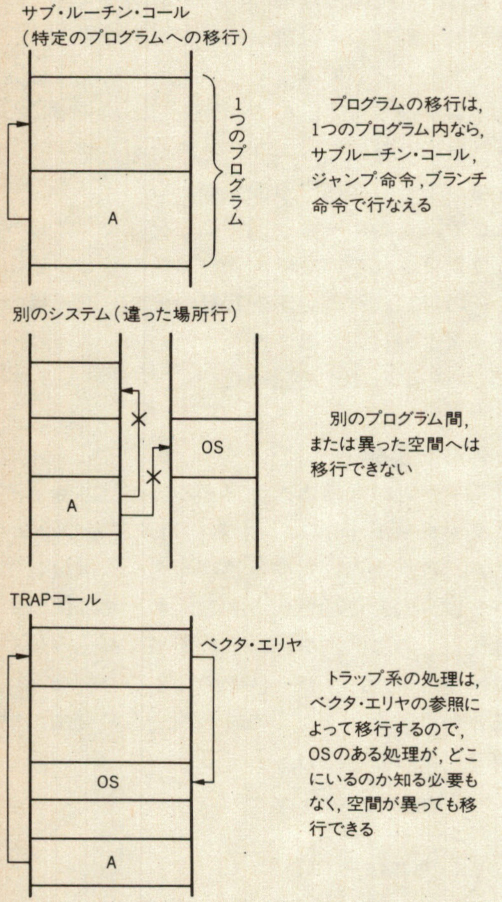
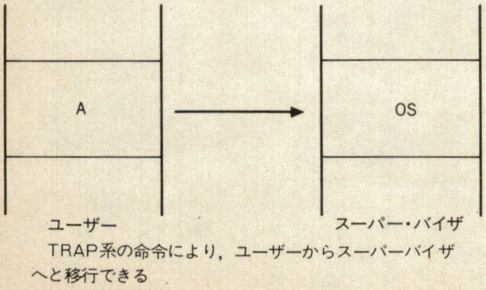


図5-2-4　TRAPの目的



5）　TRAP等のソフトウェア割込み時

ソフトウェアの割込みの主たる目的は，ユーザー側あるいはスーパーバイザ側，つまりＯＳ内部からでもＯＳにあるサービスの要求を出す事である（図5-2-4）。これとよく似たものに，サブルーチン・コールがある。サブルーチン・コールは，あるプログラムが別のプログラムの助けを借りるために，ＭＰＵの実行を別に移すという点ではＯＳコールとまったく同じである。ただしサブルーチン・コールでは行き先のプログラムのアドレスが相対的にせよ絶対的にせよ，あらかじめ判っている必要がある。また同一のメモリ空間に行き先が存在する必要もある。これはＯＳの機能を利用する場合には大変不便であり，時により不可能である。なぜならＯＳは異なったシステムの環境に対応して各々作成され，プログラム・サイズも異なるため，特定の番地が決まるわけではなく，またプログラム保護のためにページングやセグメンテーションで区切られていれば，そのＯＳのプログラム・コード自体は，特定のプログラムからは見えないからである。特にユーザー側からスーパーバイザ側は各種の保護により直接には移行できない場合がほとんどである。

そこで，ソフトウェアで発生する割込みなら，各ソフトウェア割込みが独自のベクタ・エリヤを持ちこのベクタ・エリヤはＯＳ内のものであるため，システムの変化に対応してＯＳ自体が書き直し，同じソフトウェア割込みに対しては同じ処理をとる様にＯＳ側で保証する事ができる。ユーザー側のプログラムは何の変更もなく，同じＯＳが動いていれば違ったシステムの上で動く事ができるわけである（図5-2-3参照）。

またＯＳは先に述べた入出力の例の様に，ある時間がかかるタスクの必要が生じた場合で，しかもそれがＭＰＵの機能を連続的に使用しない場合にはユーザー側のプログラムを先に進めることができる。

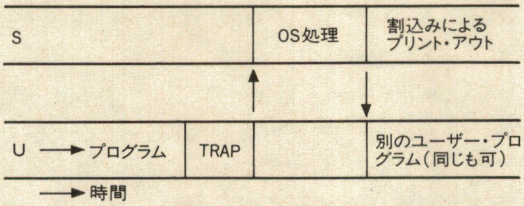
例えば，あるデータを処理して，何行分かの出力

をプリンタに出力しようとする時，ユーザー側プログラムは，そのサービス・コールをＯＳに要求し，データをＯＳに引渡せば実際の出力の実行が終わらなくても次の処理に移行できる。いちいち出力が終わるのを待ったり，場合によっては装置の立上げをしたりという待ち時間は一切関係なくなる（図5-2-5参照)。また，別の例であるが，何らかの処理をコンソールからの入力指示によって決めるなどという場合は，プログラムはＯＳがこの入力データを渡してくれるまで何もする事がない。自分のプログラムは，ＯＳがデータを揃えてくれるまで待期モードに入り，ＭＰＵの実行権を他に引き渡し，ＯＳはデータが揃うまで別のユーザー・タスクやＯＳタスクを実行させる。

そして予定している入力があったら，待期モードにあるプログラムを実行モードに移し入力されたデータを渡して続行させる(図5-2-6，図5-2-7参照)。

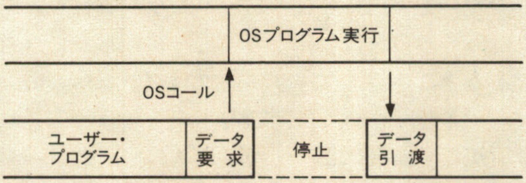
この他，ソフトウェア割込みは，データのチェックやオーバーフローにより起こされるものもある。しかし基本的には，これらのデータはＯＳを通じて与えられ，データの正当性，演算範囲の逸脱を起こす様なデータは，入出力装置の異常またはＯＳがサポートしている他のサービス・プログラムの異常と考えられる。このようなことに対する処理の第一次はＯＳが担当する様にこれらの特殊ソフトウェア割込み命令が存在する。これらの命令は，もしもあらかじめソフトウェアでチェックし限定すれば，必ずソフトウェア割込みの発生ができなくなる様にする事ができるため，ユーザー側で処理できるエラーチェックなら，その様に自己のプログラム内部で処理し自己内では処理できないＯＳを通じて他からやってくるデータに対しては，チェック＆ソフトウェア割込み命令を使用し，処置をＯＳに返してやることがこのような場合の使い方である。

図5-2-5　マルチ・タスク下でのプリントアウト例



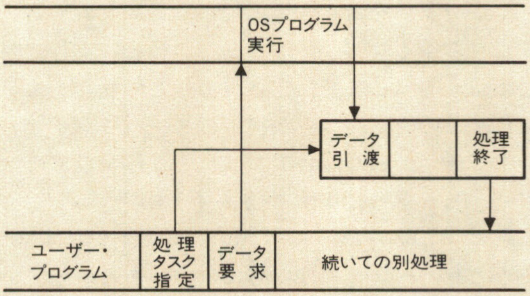
ユーザー・プログラムは入出力要求（この場合プリント・アウト）が生じると，データを用意してTRAP命令を行なう。すると処理はＯＳ(スーパーバイザ)に移り，ＯＳは必要なデータを集め入出力装置に起動をかける。そして印刷装置へ，データを送る（この時，一般に割込み処理が使われる)。割込みの間には別の処理が行なわれ，マルチ・タスクとなる。

図5-2-6　入力待ちの例



ユーザー・プログラムは，処理すべきデータが必要となった時，ＯＳコールによりＯＳのサポートを要求し，自己の処理は停止して，他のプログラムに実行能力を引き渡す。ＯＳは要求されたデータが揃うと，先に中断したタスクを継続させる

図5-2-7　処理要求と継続先指定



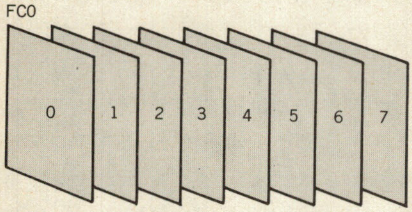
ユーザー・プログラムは，ＯＳに何かの処理，サポートを要求して，自分はその処理待ちの間，また別の処理を実行する。ＯＳはデータが揃うとユーザー・プログラムの指示されたプログラムにサポート結果を渡し，そのプログラムを実行せしめる。そのプログラムは実行が終れば，自己の別の処理に移行する。

5-3　オルタネイト・ファンクション・コード機能

ＭＭＵまたはメモリ側でＦＣコードをデコードしているシステムでは，ＯＳ側にいる時は，ユーザー側のファンクション・コードが出力されないため，ユーザー側のメモリをアクセスできない点を補なうものとして，68010以上のＭＰＵに付加された。

この機能は，専用の代替ファンクション・コードを入れるレジスタ二つと，このレジスタを使ってメ

図5-3-1　最大八つのFC空間を持つ



モリ・レジスタ間の転送を行なう専用の命令から構成される。

ファンクション・コードを入れておくレジスタへのアクセス，またこれを使ったメモリへのアクセスの命令は共に特権命令である。

手順としては，まずメモリから読出そうという場合は，オペランドのソース側を代替コードでアクセスするので，Source FCレジスタ，〈ＳＦＣ〉，メモリへの書込みの場合には，オペランドのディストネーション側などでDistonation FCレジスタ〈ＤＦＣ〉，へ「MOVEC」命令を使い，代替しようとするファンクション・コード＝ＦＣ，３ビットを記入しておき，次に「MOVES」命令でその空間のメモリに読書きを行なう（図5-3-1参照）。

この機能により，ＭＰＵはユーザー・プログラムやユーザー・データの変更，修正，またはデータの引渡し，引受け等をMMUやＦＣデコーダを処理せずに自由にできる事になる。

68000ファミリでは，以上の様にＯＳの機能は，ＯＳプログラムに専有させるため，専用のスタック，ＦＣ２という専用のアドレス線を持ち，割込み，例外処理は自動的にスーパーバイザへ移行する，更にシステムの状況を変更する，またユーザー・モードへ移行する処置をとるといった命令は，すべて特権命令としてユーザー・モードでの使用を禁止し，ユーザー・プログラムが誤ってシステムの状況を変更する事のない様に，またユーザー・プログラム側ではエラーが，システムに重大な影響を与えない様にするため，各種のチェック＆トラップ命令を設け，ユーザー・プログラムとＯＳをはっきり区別して，システムが作れる様になっている。

もちろん小規模のシステムは，全てスーパーバイザで組んでも作れるが，従来の８ビットの手法と同じなため68000ファミリの特徴を生かしていないという事になる。

5-4　マルチＯＳ

68000ファミリでは，68010／20／30の機能として一つのハードウェア上で，複数のＯＳを動作させる為のサポート機能がある。

これは，68000ファミリではＯＳに対するサービス・コールを，全て例外処理という統一した方法で処理しているため，この例外処理を起動するベクタテーブルを複数持つ事ができるならば，動作させようとする各ＯＳに専用のベクタ・テーブルを持たせれば良い事になる（図5-4-2参照）。

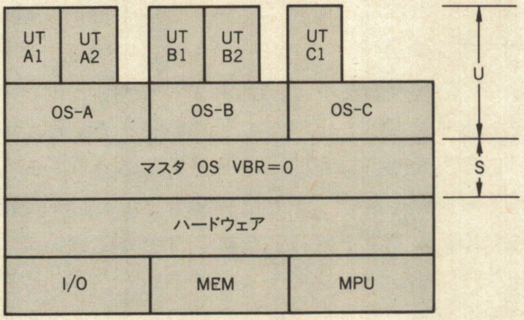
つまり，あるユーザー・タスクがあって，そのユーザー・タスクはあるＯＳ上で動作するものとして作られている時，このユーザー・タスクが実行できて，かつ，その内で各種のＯＳコールが実行されていれば，そのＯＳは動作していると言う事ができる訳である。この時，ユーザー・プログラムからのスーパーバイザ・コールは，TRAP命令や他の例外処理を引起こす命令によってなされるので，各スーパーバイザ・コールによる要求が正しく，そのＯＳ内の処理に引渡されれば良い事になる。

これを実現するには，ＯＳが２レベルに分かれる必要がある。第１レベルがマスタＯＳである。このＯＳはマルチＯＳをサポートする機能をもち，通常のハードウェアを含む入出力管理，システムの実行時間管理の第１レベル，つまりどのＯＳに実行時間を与えるか等はここで決定される。この様なシステムでは，このレベルのみが実際のスーパーバイザ・モードで働いており，各アプリケーションＯＳはユーザー・モードで動作する（図5-4-1参照）。

これは同じ68000ファミリ用のＯＳであっても，例外処理とそのベクタの使い方には差があるのでアプリケーションＯＳ向けの例外処理を一度それ専用のベクタ・テーブルで受け，その実際の処理は，マスタＯＳへ再度内部割込みにて引き渡す事ができる必要があるからである。

この時，各アプリケーションＯＳ内で行なわれる

図5-4-1　マルチOSのシステム構成



マルチＯＳシステムの場合の形態及びスーパーバイザとユーザーのMPUのモードの使用範囲

ＯＳ専用の命令は，プリビレッジ・バイオレーションの例外処理を引起こすため，マスタＯＳ側でその発生原因を調べ，ソフトウェアでエミュレーション（模擬的に実行してやる事）を行なう必要がある（図5-4-3参照）。

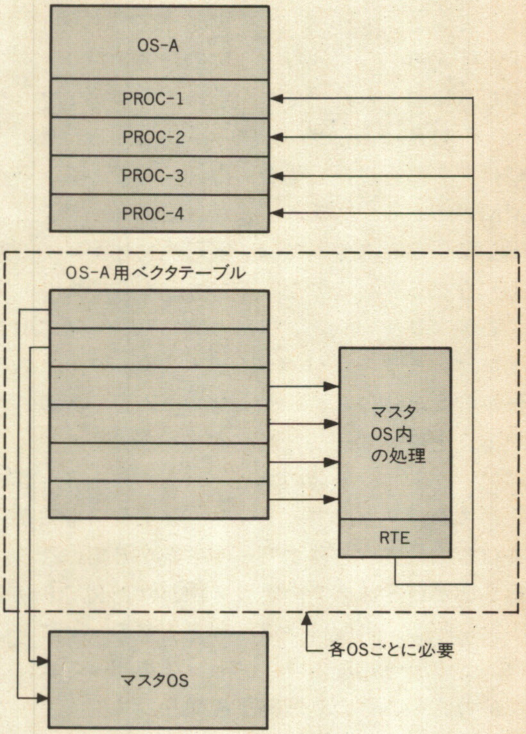
なぜ各ＯＳをユーザー・モードに置くのかと言えば，まさしくこの特権違反の例外処理にある。各ＯＳが要求するシステムの実行環境は当然異なるためマスタＯＳは各ＯＳごとに異なる内部割込みに対し，対応した処置を行なうが，これは同じサービスの要求が違った形式で与えられることであるから，これを翻訳して，各アプリケーションＯＳからの要求により各ＯＳの要求する環境に合致しているかの様にソフトウェアで見せかける作業を行なう。

したがって，各ＯＳがスーパーバイザ・モードにあって，実際に実行環境を変更できると，他のＯＳの実行環境に影響を与えてしまう事があり得るし，最悪の場合，マスタＯＳに影響を与えてマスタＯＳ自体の実行ができなくなる可能性もある。

そこで，実際にスーパーバイザ・モードで動作するのはマスタＯＳのみであり，各アプリケーションＯＳは，実際にはユーザー・モードで働かせ，実行環境を実際に変更できないように，ソフトウェアでロックをかけて行なう。

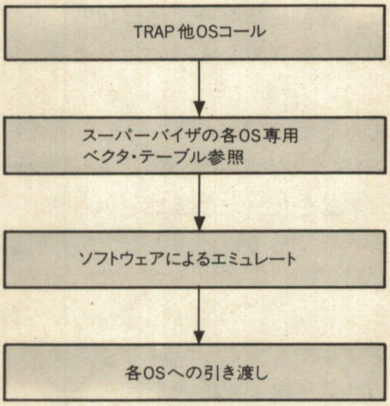
但し，各アプリケーションＯＳに実行時間を与える時，各ＯＳ専用のベクタ・テーブルを作製し，各ＯＳへのスーパーバイザ・コールの実行をすみやかに行なえる様にする。これは，スーパーバイザ・コールが起きると自動的にスーパーバイザ・モードとなり，例外処理用のベクタ・テーブルが参照されるがベクタ・テーブルをマスタＯＳが全て管理してお

図5-4-2　マルチOSの例外処理



マスタＯＳ上にスレーブＯＳ—Ａが乗って働いている時，マスタＯＳは，このＡの様な各ＯＳごとにベクタ・テーブルを用意して，スレーブＯＳ—Ａ内の例外処理（PROC1，PROC2，PROC3，PROC4，‥‥）をサポートする。この時，特権命令違反の例外処理等の様に，マスタOS側にてソフトウェア・サポートが必要なものもある。

図5-4-3　通常のOSコールの手順



りマスタＯＳ内の各ＯＳに対応した処理をベクタは指示している。

TRAP命令のベクタなど，実際に各アプリケーションＯＳに引きわたして良いものでも，各アプリケーションＯＳはユーザー・モード側にいるため，直接にはそこに行けない場合がほとんどである。それはMMUのあるシステム，またはＦＣコードをデコードしているシステムでは，実行先自体がメモリ空間上に存在しないことなってしまうからであり，実際に各アプリケーションＯＳの実行アドレスに引き渡すには，まずスーパーバイザ・スタック上にある例外処理要求時のスタック・フレームを，各ＯＳのスタック上にコピーし，対応する行き先をスタック上にプッシュし，ＳＲのコピー（もちろんＳ＝０）を更にスタック上にプッシュし，ＲＴＥ命令によって各ＯＳ側に実行を移すという手続が必要になる。各ＯＳ側がＯＳタスクの実行を終わり，ＲＴＥ命令の実行に際し，再度特権違反が発生する。マスタＯＳは，この例外処理の発生アドレスを調べる事により，各ＯＳタスクの処理終了を知り，スタック・フレームをコピーし，実際にＲＴＥを行ない元のユーザー・タスクに戻す事ができる。

この際に，通常のスーパーバイザ・コールは「スーパーバイザ・コール→マスタＯＳ空間の各ＯＳ専用のベクタ・テーブル参照→指示されたマスタＯＳ内のソフトウェア・エミュレータ実行→マスタＯＳ内で実行か，各ＯＳに直接処理が移ったかの様にエミュレートして引き渡し」の手順を通って実行される事になる。

もちろん，各ＯＳは68000ファミリ用のＯＳであり，各ベクタ・エリヤを０番地スタートで作ること，TRAP，ＲＴＥを使ったＯＳコールのサポートを行なっているべきであるなどの制限はある(**図5-4-2**参照)。

現在，コンピュータの普及に伴い様々なＯＳが提供され，ヒューマン・インターフェース重視傾向が強まり，ＯＳの機能の高級化に伴い，ユーザーのＯＳに対する習熟度が問題となるので，ユーザーが自分の使いたい，または使いなれたＯＳを使えるマルチＯＳシステムも増えてくるものと思われる。

5-5　仮想記憶について

ここに言う仮想記憶 (Virtual Memory) とは，実記憶を物理的にＣＰＵのバス上に存在している記憶部とし，これに対し実際には存在しないのに，あたかもメモリが存在するかの様な環境をＣＰＵに提供して，各タスクまたは各プログラムが実記憶の制限にとらわれない様なシステムの事である。この様な環境下では，プログラマはＣＰＵ固有の全アドレス空間にメモリが実装されているものとして，プログラムを作製する事ができる(**図5-5-1**参照)。

つまり，プログラム自体の大きさも，プログラムが扱うデータ量も，ほとんど気にする必要がなくなる事，他のユーザーとのメモリの分け合い等の心配がない事，プログラム・コードの発生時，効率の良いベース・ページの部分が使える事等，ソフト的には良好な環境が与えられる。

欠点としては，システム自体が複雑になる事，実行時間が遅くなる事，ＯＳが複雑になる事，比較的高価である事などがある。

この様なシステムは，どのようにして実現されるか。単純に言えば，実記憶のメモリの替りをハードディスク，または他の同様な外部記憶装置に実行させる事である(**図5-5-2，3**)。

外部記憶は，ＣＰＵが直ちにアクセスする事はできないので，ある程度以上の実記憶も必要である。

また当然の事であるが，この実記憶は分割されてＯＳの使用部分，ユーザーごとの各タスクに割当てられる部分に分かれるので，あるタスクが働いている時に，他のタスクで使用されている実記憶に影響を与えないように実記憶に存在する所と外部記憶に存在する所を区別するために，メモリ管理装置（ＭＭＵ）が必要となる。更にこのMMUは，実記憶のアドレス（物理アドレス）と実行アドレス（論理アドレス）の整合も行なう。また外部記憶と実記憶の入れ替えを頻繁に行なうため，メモリへの直接転送を行なう部分（ＤＭＡ）も必要となる(**図5-5-4**)。

この様子をＣＰＵから見ると，外部記憶のディスク上にあるプログラムやデータを，実記憶というス

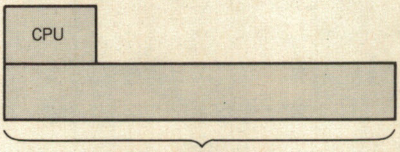
クリーンに投影して，この投影像を元にＣＰＵはタスクを実行するという事になる(図5-5-5)。

このようなシステムに要求されるＣＰＵの特別な性能とは何であるか。それは，まず第一に，「あるメモリにアクセスを行なった時，アクセスができなかった」という事を認識できる事である。これは，外部記憶から実記憶に写されている部分以外にアクセスが行なわれた，つまり外部記憶域中のデータやプログラムをアクセスした事になり，そして外部記憶から実記憶への転送，MMUの設定の仕直し，場合によれば実記憶から外部記憶への転送を要求し，全体としてアクセスに失敗した部分をアクセスできる様にする必要が生じたという事を認識できる回路構成が必要である。

この為には，DMACやディスクを向いたり，ＣＰＵを向いたりする専用のデータ・ハイウェイを持つメモリ・システムを付ける場合もある。

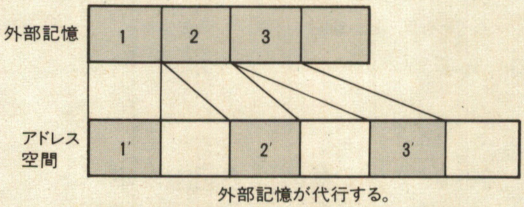
更に，以上の様な補正が行なわれ該当部分がアクセス可能になったとして，ＣＰＵはそのメモリ・サ

図5-5-1　仮想記憶の長所



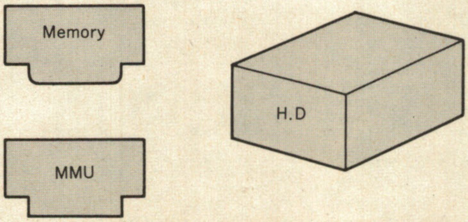
仮想記憶は，全アドレス空間にメモリが実装される様に見える

図5-5-2　仮想記憶の記憶部



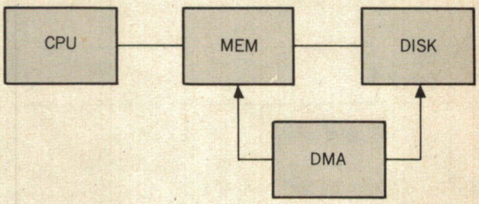
仮想記憶上のメモリの内容は，外部記憶装置が代行する

図5-5-3　仮想記憶の条件



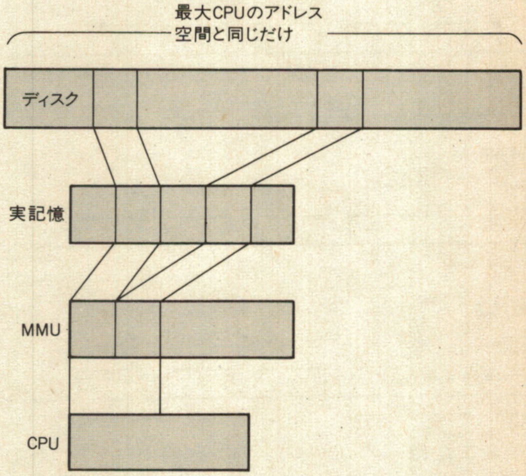
仮想記憶はメモリ（主記憶），MMU，ハードディスク装置（外部記憶）が必要である

図5-5-4　仮想記憶のサポート



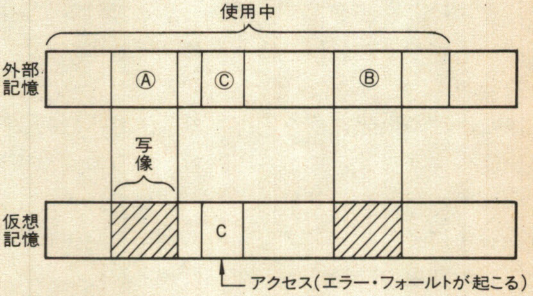
主記憶と外部記憶の間のデータやプログラムの転送は，DMAにて行なわれる

図5-5-5　ディスクからのアドレス変換



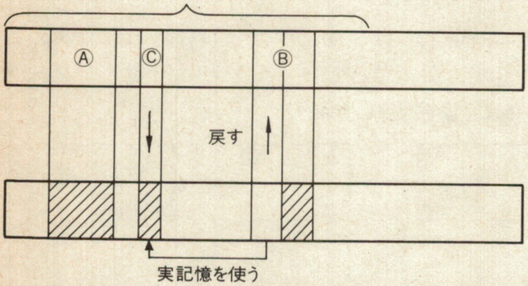
CPUは，MMUによってアドレス変換されたメモリの内容を扱う。このメモリの内容は，外部記憶の内容のコピーである

図5-5-6　実記憶不在の検出



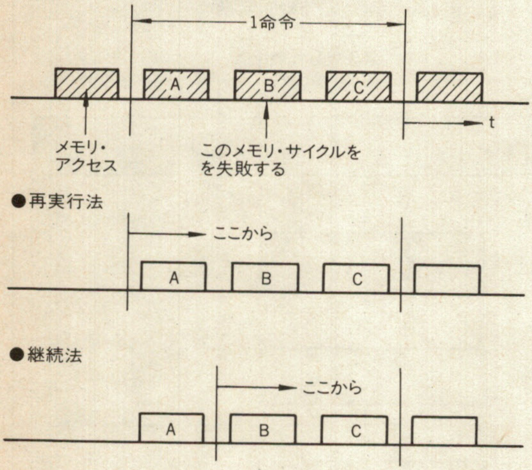
仮想記憶のある点（C）をMPUがアクセスした場合，外部記憶上にコピーが存在しても，主記憶に移されていなければアクセス・エラー（バス・エラー）が起こる

図5-5-7　ディスク―メモリ間のスワップ



先のエラーの処置として，アクセスした内容が外部記憶中に存在すれば，主記憶中の一部をディスクに戻し，空いた主記憶にアクセスして，エラーが発生した部分のコピーを外部記憶より移して再構成する

図5-5-8　再実行法と継続法



イクルまたはその命令サイクルを再実行できなければならない。補正された実記憶をアクセスして，正常にプログラム実行が継続できて始めてシステムとして有効となる訳である。

仮想記憶を実現するＣＰＵとしては，中位ミニコン以上はほとんど全てこの機構を持っている。逆に８ビットのマイコンは，ほとんど全てサポートはできない。

また，この仮想記憶を実現する手法で一般的なのは，次のようなものである。即ち，ある命令を実行する直前の状態をＣＰＵ内にセーブしておき，１命令実行中にメモリ・アクセスのエラーが発生して，仮想記憶利用の為に，何らかの再設定が行なわれた後にこの命令実行直前の状態を改めてＣＰＵにセットし，そこから再度実行する命令再実行法と言われるものである。この方法は，ＣＰＵ内の全てのレジスタを２重にし，１命令終了ごとにＣＯＰＹして同じとし，命令中断後の再実行時には，コピーからマスタへと内部データを移す事により行なう。

もう一つの方法は，命令実行の流れによるＣＰＵ内の状態をスタック等にセーブしておき，補正後そのメモリ・サイクルから実行するもので，命令継続法と言われる。モトローラの68000ファミリは，この命令継続法を採用している。

この二通りの方法には一長一短はあるが，それは特殊な状況においての事で，ほとんどの場合どちらの方法でも問題はない(**図5-5-8**)。

仮想記憶のサポート手順としては，いまＡとＢの部分が実記憶に写像となってプログラムが実行されMMUによりその時点ではＣＰＵのアドレス空間中ＡとＢの写像部分だけに実際のメモリが存在する様になっていた(**図5-5-6**)。

そこに，Ｃの部分をＣＰＵがアクセスし，メモリのアクセスの失敗(メモリ・フォールト)が発生した。

システムは現在実行中のプログラムの実行を一時棚上げして，Ｂの実記憶の一部を外部記憶に戻し，Ｂに対するMMUが割り当てているメモリの領域（メモリ・セグメント）の変更を行ない，新たな領域を設定して，ここに先にＢから取り上げた実記憶を割り当て，そこへ外部記憶からＣの相当部分をコピーし，Ｃの部分のアクセスができる様にして（**図5-5-7**）再度アクセスを行なわせる。

5-6　68000ファミリの仮想記憶サポート

68000ファミリでは，仮想記憶をＭＰＵ自体でサポートする機能を持つ68020／30／12／10のグループと，外部からの補助がなければサポートできない68000／08のグループに分かれる。

サポートできる68010以上のＭＰＵの場合には，ＭＰＵはメモリのアクセスに失敗した場合，内部のコントロール状態とＰＣ，ＳＲの内容を，スーパーバイザ・スタックにセーブする。内部の各レジスタの内容については，ソフトウェアで対処する(MOVEM命令)，セーブする所は必ずしもスーパ

ーバイザ・スタック上とは限る必要はない。

68020以上では，この時，最長46バイトの内部情報（内部レジスタの内容ではない。）が待避される。

その後，ＭＰＵ自身の実行によりメモリの補正を行なって，中断した命令を継続すれば仮想記憶のサポートができる（ＲＴＥ命令，ロング・フォーマット）。

68000で仮想記憶をサポートするためには，自分自身で中断した命令を継続実行する事はできないためＭＰＵの機能のうち，失敗したメモリ・サイクルのみを再試行するリトライ機能を使う。メモリのフォールトが起きると，外部回路によりＭＰＵをホルト（HALT：一時停止）の状態にし，他のプロセッサで外部記憶と実記憶の入れ替えを行ない，その後，HALTを解除する事によって，止めておいたメモリ・サイクルを行なわせる事によって実現している。

この方法は，仮想記憶サポートのために，もう一個の実際の仕事をしないＣＰＵが必要だという点では非能率であり，外付回路も大きくなる。したがって，68000とピン・コンパチブルで仮想記憶をサポートできる68010がある以上，これから設計される装置ではこの方法の必要はない（**図5-6-1**）。

このサポートの手順をあげる。

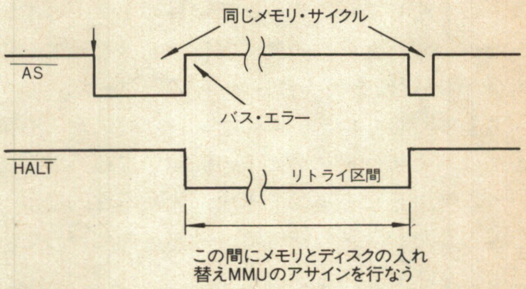
68020／30／12／10では，命令の実行途中にメモリのアクセス失敗が起き命令の実行が中断されると，ＭＰＵの内部状態をシステム・スタック上に待避しバス・エラーの処理を実行する。

このバス・エラーは，メモリのアクセス・サイクルの時間を外部で測り，一定限度内にアクノレッジが返ってこない場合，メモリのアクセスに失敗したものとして，ＭＰＵのバス・エラーのピンに与えられる信号によって起こされる例外処理である。

又，MMUは設定外のアクセスを検知して，やはりバス・エラーを発生させる。この符号によりＭＰＵは異常アクセスの処理に移行する。

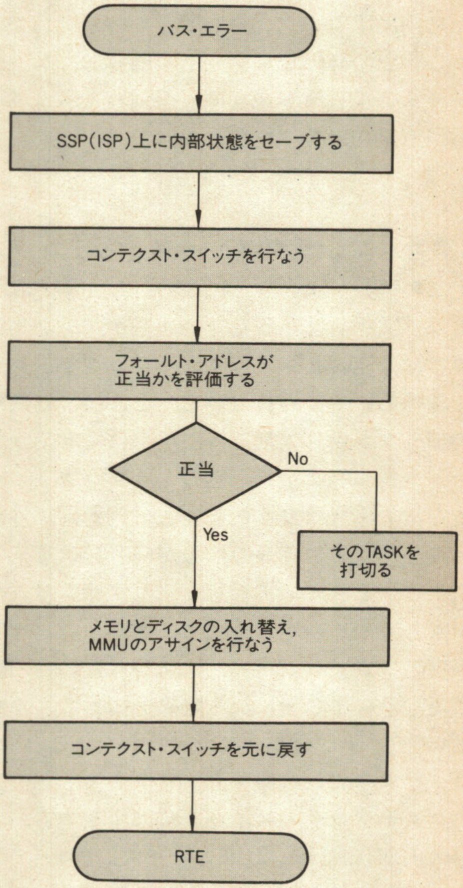
バス・エラーの処理の中で，ＭＰＵはソフトウェアの処理により，フォールトの発生したアドレスを評価し，そのアドレスが実行中のプログラムにとり妥当である場合，つまり外部記憶のディスク内に写像があると判断すれば，そのディスクからデータなりプログラムを実記憶に転送し，MMUにより該当するアドレスに主記憶のアドレスを返還して中断した命令を再実行させる（**図5-6-2**）。

図5-6-1　68000での仮想記憶



68000 で仮想記憶を作る方法として，リトライを利用してエラーの発生したメモリ・サイクルを中断し再実行させるが，このHALT区間中に別のMPU によってシステムの再構成を行なう

図5-6-2　一般的なバスエラーに対する処理手順



評価した結果が不当な場合には，そのプログラムは予め予約しているデータ領域外のアクセスを行なったか，プログラムの異常実行が起きたかで，この場合,例外処理を扱うＯＳは,その異常を起こしたタスクを中断してエラー情報をユーザーに知らせる。これはユーザー・タクス上の問題で，ＯＳとしては不当の理由を知る事はできず，また本質的なエラーを起こしたタスクの継続実行は不可能だからで，継続は無意味でもある。

システム・スタックの上にセーブされる内容は，68010／12と68020／30では異なる。これは68020／30では実行中断の場合と，実行と実行の間でフォールトが起きた場合を区別している事と，内部の改良のため内蔵するレジスタが増えているためである。

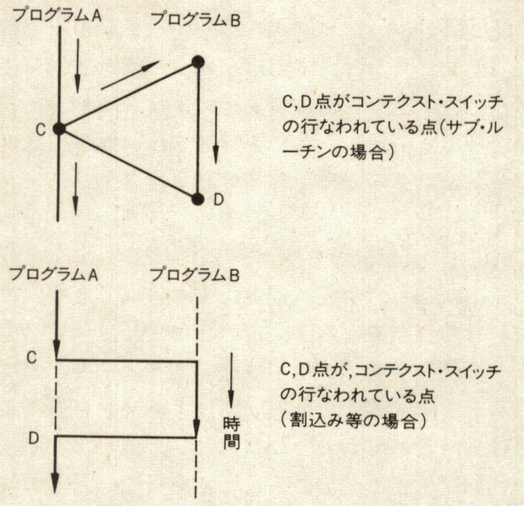
この点の詳細なスタックの内容は，ユーザーズ・マニュアル（第６章５項）か，バス・エラーの項を参照（第10章３項）されたい。

〈参考〉コンテクスト・スイッチング

このコンテクストというのは，文字どおり直訳すれば「文脈」という意味であるが，コンピュータ関係では，連続して実行されるプログラム列という様な意味あいで使われている。これは，外部割込み，内部ソフトウェア割込み，場合により，サブルーチン・コール等により，実行するプログラムが切り替えられる場合に良く使われる。例えば，外部から割込みやバス・エラーの信号がＭＰＵに与えられるとＭＰＵはそれまで実行していたプログラムの実行を中断し，割込み処理やバス・エラーの処理といった別種のプログラムの実行に切替わる。この時，また各処理が終了し元に戻る時等を指して，コンテクスト・スイッチが行なわれたと称している。同様に各プログラムの流れ自体をもコンテクストと言う。

コンテクスト・スイッチ時に注意するのは，ＣＰＵの内部状況,周辺状況のセーブ,リストアである。これは，一般の割込み時など，必ず元のプログラムに復帰して，正しく動作を継続できなければならないからである。ＭＰＵの内部レジスタやMMU等,実行プログラムの切換えに伴なって，設定を変更する恐れのある部分を退避させ，新たな設定を行ない新しいプログラムを実行する（**図5-7-1**）。

図5-7-1　コンテクスト・スイッチの様子

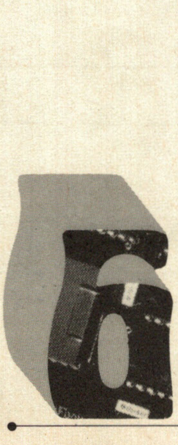


ところが，この退避動作自体を実行する事が，ソフト的には既に新しいコンテクストの実行になるため，切換えのトリガとなる事象の発生時に，この事象の発生により即時に変更させられる可能性のあるレジスタは，最低限ＭＰＵの機能として持っていてくれないと困る。８ビットの6800マイコンでは，割込み時には全レジスタの自動待避をしてくれたためこの様な配慮とは無関係であったが,68000ファミリの様に内蔵レジスタの数が多くなると，全部を自動的に退避するのは，不合理かつ不必要であり，最低限のＰＣとＳＲのみが退避される（特別の例外処理ではＭＰＵ内の情報も退避させるが，プログラミング・モデルからはこの２種のみである)。他のレジスタは必要に応じてソフトウェアの処理で退避する。これは「MOVEM」１命令で実行できる。

これ以外で特に注意するのは,68020／30の内蔵キャッシュ・メモリで，MMUのあるシステムの場合コンテクストが切換わった時も，ほぼ同一アドレスを実行し，そのため古いキャッシュの内容を読出してしまう事態が考えられるため，スイッチする時はキャッシュ・クリヤを行なう必要がある。

第6章

68000ファミリの機能内容



6-1 データ・レジスタ

68000ファミリは現在，どのMPUも32ビットのデータ・レジスタを８本持っている。このレジスタはＡＬＵと結びついているため，ほとんどの論理演算，数値演算の場合に，各命令はオペランドとして，演算側，被演算側のどちらか一方はデータ・レジスタを要求する。

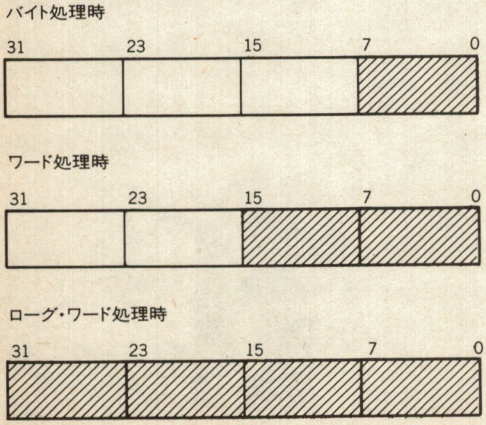
処理語長は１バイト，２バイト＝ワード，４バイト＝ロング・ワードと豊かであるが，68020／30では更にビット・フィールド系の命令がメモリとのやりとりに使われる(図6-1-1)。ビットを除き(ビットは番号で指示される）データ・レジスタ内の内容を変更した場合，下位の処理語長に当たる部分だけが変更され，上位の非処理語長の部分には変更なく前の内容が保たれる。

また，このレジスタはアドレス作業を行なう際に汎用のインデックス・レジスタとして使用できる。68020／30ではベース・ポインタ，アウター・ディスプレイスメント・レジスタとしても使える。

その他，命令の実行により自動的に１減算が行なわれて，範囲と条件が，同時にチェックされる「DBcc」命令，68020／30では数値（０～９)を，アスキー文字やEBCDIC文字に直す命令，他の内蔵レジスタとの交換，レジスタの上位拡張等，強力な命令はこのデータ・レジスタにより実行される。その時８個のデータ・レジスタはまったく等価で何ら差異がないのが特徴である。

このレジスタでオフセット値や，数値を扱う時に一つ注意する点がある。それはデータ・レジスタ内では数値はＢＣＤを省きオフセット・バイナリとして扱われる事である。この為にレジスタの演算長を合致させる目的の「ＥＸＴ」命令がある。オフセット・バイナリとして扱うと問題のある演算（乗除算）では，サイン（＋／－有）とアンサイン（±なし，＋のみ）の２種の命令がサポートされている。

図6-1-1　データ長の指定によるデータ・レジスタの有効となる範囲



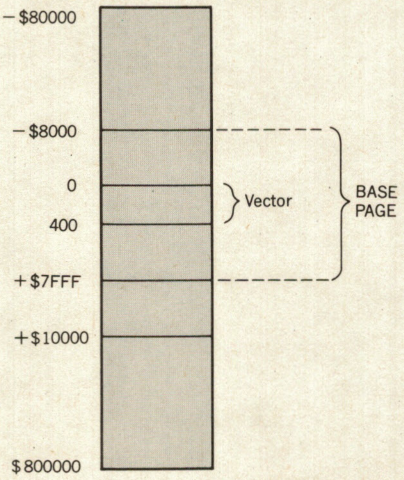
6-2 アドレス機能

アドレス機能はＭＰＵ内部で32ビット長を持っているが，外部に出力されるのはＭＰＵのタイプにより異なっている。

表6-2-1　アドレス線数

CPUタイプ	アドレス記号	線数	幅
68008	A０～A19	20本	8ビット
68000	A１～A23	23	16ビット
68010	A１～A23	23	16ビット
68012	A１～A30	30	16ビット
68020/30	A０～A31	32	8ビット

図6-2-1　68000シリーズMPUのアドレス空間におけるベース・ページの位置



68000シリーズMPUのアドレス空間におけるベース・ページの位置。ベース・ページは短縮型のアドレス・モードが使える。

どのタイプでも各アドレスは各々１本の専用出力ピンを持っている。またリニア・アドレスをサポートしているため，64Kワードの制限はなく，「$0FFFF」のアドレスは単に１加算するのみで「＄10000」になる。もちろん，プログラム・コードや，データが64Kワードの境目にかかっていても問題はない。

68000ファミリではアドレスはバイト単位で付けられている。またアドレスの計算はオフセット・バイナリで正負を含んで計算されるので，アドレス空間のイメージは，図6-2-1の様にも考えられる。

この時，０を中心に，16ビットでアドレスできる－＄8000～＋＄７ＦＦＦをベース・ページと言う。

このアドレス・バスは単にメモリのアドレスを指示するのみでなく，外部割込み時には受付けた割込みの優先順位レベルを表示し，68020／30ではコ・プロセッサの指定にも使われている。

6-2-1　68000のアドレス・ラインＡ０

68000／10／12では，Ａ０が直接には出力されておらず，ＤＳ(データ・ストローブ)との合成信号となって出ている(図6-2-2)。

このＤＳはＬＤＳ，ＵＤＳの２本が用意され，16ビット・バスを８ビット×２のバスに分割し，単一バイトのアクセスを許し，かつ両方同時のアクセスもできる様にするためである(図6-2-2)。この信号は，下位ビット側（Ｄ０～Ｄ７）はＬＤＳ，上位ビット側（Ｄ８～D15）はＵＤＳと名付けられ，バイト・アクセスの時は偶数アドレスならＵＤＳ，奇数アドレスならＬＤＳのみが出力され，ワード，ロング・ワード・アクセスの時は両方が出力される（図6-2-3)。

6-2-2　68020／30のアドレス・ラインＡ０

68020／30では，Ａ０はＡ０として出力され，サイズ・コードといって，何バイト分をアクセスしているのかを示す信号線が別に用意されている。これによってメモリ側でＵＤＳ／ＬＤＳのような物を作って対応する事になる。68020／30のバスの幅は32ビット一定ではなく，1バイト，2バイト，4バイトと切替えられるため，ＬＤＳ／ＵＤＳの様な一定の幅に対応した信号を作る事ができないためである。

図6-2-2　アドレス空間に対するアドレスの数え方とLDS，UDSの関係

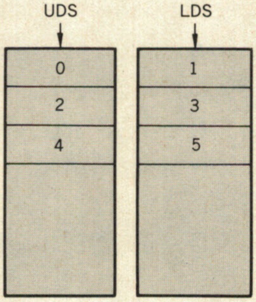


図6-2-3　UDS，LDSとA0の関係

バイト・アクセス時

	$\overline{\text{UDS}}$	$\overline{\text{LDS}}$
A0＝0	0	1
A0＝1	1	0

ワード，ロング・ワード・アクセス時

	$\overline{\text{UDS}}$	$\overline{\text{LDS}}$
A0＝0	0	0
A0＝1	禁止	禁止

6-2-3　メモリ・デコーダ

メモリ側では各ＭＰＵのタイプに合ったデコード回路が必要である。アドレスはバイト・イメージで付けられているため，バスの幅がバイトならＡ０以上を，ワードならＡ１以上を，ローグ・ワードならＡ２以上をメモリに与え，メモリ・ブロックが受入れる分はそのままメモリに与え，上位はデコードする。

メモリの幅に相当する信号分，例えば68020／30でロング・ワードを使用した時のＡ０，Ａ１もデコードされ，各バイト単位のストローブとして与える。

68000／10／12では，この部分はＭＰＵに内蔵されて，ＬＤＳ，ＵＤＳという信号になっている。

68020／30では，デコードした信号に何バイトのメモリを使うかを示している信号ＳＩＺ０，ＳＩＺ１を合成しＤＳでストローブしてメモリに与える（図6-2-4)。68020／30ではバスの幅が一定とは限らないため，このデコードは各メモリ・ブロック側にて行なう必要がある。

更に68000ファミリでは，システムによってはメモリ側でファンクションコード（ＦＣ）のデコードを行なわせる場合がある。

代表的な例は，ＯＳマシン用にＯＳが扱う入出力（メモリ・マップドＩ／Ｏなため，メモリとしてデコードする）の場合である。アクセスは必ずスーパーバイザ状態（＝ＯＳ・モード）のデータ・スペースとなるので，ＦＣ２＝Ｈ，ＦＣ０＝Ｈの条件が必要であり，更にこれを割込みアクノレッジ等で使われるＣＰＵスペース（ＦＣが全部Ｈ）と区別するため，ＦＣ１＝Ｌも含める必要がある場合もある。

6-2-4　ＦＣのデコード

メモリ側では，MMUがあり，MMUでＦＣにより，各スペースの振り分け，アドレス変換を行なって単一のメモリ空間を分割して各スペースに専用のページ，またはセグメントとして与える場合は，ＦＣのデコードはしない(**図6-2-5，6**)。

しかし，ＯＳやスーパーバイザ・スタック保護用にスーパーバイザ専用のメモリを設けるために，ＦＣコードをデコードしたメモリを付ける事はよく行なわれる。

ここで注意したいのは，このＦＣコードのデコードと，アドレス上位のデコードとの関係である。68000ファミリでは，一般に入出力はアドレス値の最大付近に置いている。これは，リセット時ベクタ・テーブルが０番地にあり，そこからアドレス値の増加方向にＲＡＭを置いていくが，ＲＡＭの量の制限を少なくするためには，最上位付近にＲＯＭやＩ／Ｏを置くのが合理的なためである(**図6-2-7**)。

これらのＲＯＭやＩ／Ｏは，一般にスーパーバイザ側に，かつ上位のアドレスに常駐している。

ところが，外部割込み時に外部から割込みベクタを読むための割込みアクノレッジ・サイクルを考えてみると，Ａ４以上の全てのアドレス線はＨになっており，最上位のアドレス付近を示している。

このため，デコーダには注意を払って設計する必要がある。上位アドレスのみをデコードすると，この割込み発生時と同じ上位アドレスが全てＨになる所を使う事になり，この時に割込みアクノレッジ・サイクルで割込みアクノレッジと通常のＩ／Ｏアクセス・サイクルが同時発生の形になり，システムに誤動作をさせエラーに導く怖れがあるためである。

図6-2-4　ＳＩＺ０；ＳＩＺ１を各メモリに与える

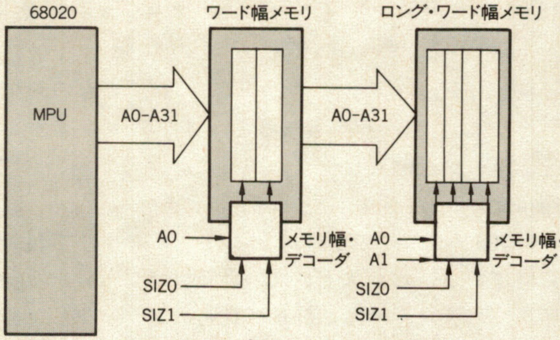


図6-2-5　ファンクション・コードとその意味

FC2	FC1	FC0	
0	0	0	
0	0	1	ユーザー・データ
0	1	0	ユーザー・プログラム
0	1	1	
1	0	0	
1	0	1	スーパーバイザデータ空間
1	1	0	スーパーバイザプログラム空間
1	1	1	CPU空間

図6-2-6　ファンクション・コードをアドレスと考えた時のアドレス空間

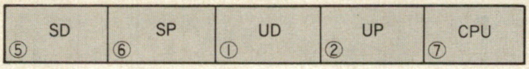


図6-2-7　一般的なメモリ空間配置(ＦＣのデコードをしない場合)

図6-2-8　割込みアクノレッジ・サイクル時のアドレス・ラインの状態(上)，I/Oアクセス時のアドレス・ラインの状態(下)

割込みアクノレッヂ・サイクル

FC			A					
2	1	0	31	30	29	28	-----	4
H	H	H	H	H	H	H	H	H

I/Oアクセス・サイクル

FC			A				
2	1	0	31	30	29	28	-----
H	L	H	H	H	H	H	X

また，システムによっては，この部分にメモリを持つものもあり得るので，ＦＣはデコードする場合には3ビットを完全にデコードする事をすすめる。又上位アドレスが全てＨになる最高位アドレスのデコード出力は使わない方が良い(図6-2-8)。

6-2-5　アドレス・レジスタ

このアドレス・レジスタは，Ａ０～Ａ６と，その時のスタックと同じ意味のＡ７があり，Ａ７はＳＳＰ（スーパーバイザ・スタック・ポインタ）とＵＳＰ（ユーザー・スタック・ポインタ）が切替るが，68020／30ではＳＳＰがＩＳＰ（割込みスタック・ポインタ）とＭＳＰ（マスタ・スタック・ポインタ）とに更に切替る。つまり，68000／08／10／12には，アドレス・レジスタは９本，68020／30では10本ある。これらは，メモリのアドレス作業を行なうのに用いられ，更にレジスタ間接アドレスに用いられ，又，インデックス・アドレッシング時に，ベースとしてもインデックスとしても用いられる。更にメモリ・アクセスごとに自動的に加算，減算する事もでき，スタックやヒープ，テーブル・ポインタとして用いられる。もちろん，メモリとの内容のやりとり（ただしワード，ロング・ワード）や，他の内蔵レジスタとのやりとりもできる。

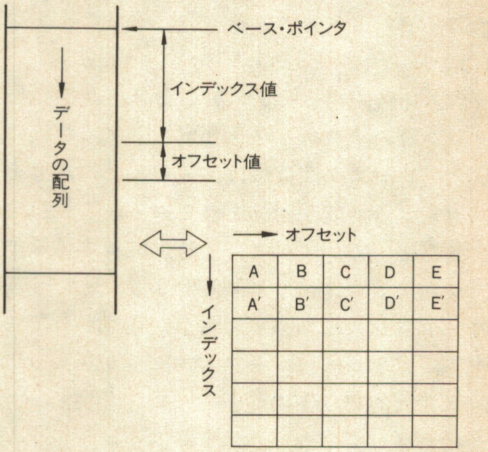
68020／30では，メモリ間接のベースとしても用いられる(メモリ間接は68020／30のみ)。

6-3　68000ファミリのアドレス方法

ＭＰＵがアドレス空間中のどのメモリを読み書きするかを決める方法，これをアドレス・モードと言うが，ここで決められるアドレス自体を〈ea〉エフェクティブ・アドレスといっている。これを決めるのに68000ファミリで最も良く用いられる方法が，アドレス・レジスタを様々に使ったアドレス・モードである。

一番単純なのは，32ビットのアドレス・レジスタの中に，アドレスの値そのものを入れ，この値の示す所をアドレスする方法である。これは，このアドレス・レジスタの内容をそのままアドレス・バスに出す事によって行なわれる。更に，このアドレス・レジスタの内容は変えずに，アドレス・バスに出力するデータに，ある一定の値を加えたり，他のレジスタの内容を加算することもできる。このような場合，元の点を示すアドレス・レジスタをベース・ポインタと呼び，あるデータの基点を与えるものとしそこから何番目のデータであるかを示すレジスタをインデックス(索引)レジスタと呼んで区別する。

図6-3-1　オフセット・インデックス付アドレスレジスタ間接アドレス・モードにより構成されるデータ配列の理的な意味(上)と考え方(下)



68000ファミリでは，原則としてアドレス・レジスタとプログラム・カウンタはベース・ポインタになれるが，データ・レジスタはなれない。インデックス・レジスタには，アドレス／データの全部のレジスタがなる事ができる(図6-3-1)。

この様なアドレス・レジスタを使いアドレスを行なう方法を総称して，アドレス・レジスタ間接と言う。これは，プログラムはアドレス・レジスタを指定して間接的にアドレスを指示する事からきている。

ちなみに，ＭＰＵの命令コードに対象とすべきメモリの番地を直接指示する場合は，絶対アドレスという言い方をする。

また，アドレス・レジスタ間接に似たものにメモリ間接がある。これは，メモリ中にアドレス・テーブルを作り，プログラムはこのメモリ・テーブル内のどれかをまず指示し(この時，68020／30ではアドレス・レジスタが使われる)，そこから読み出された

アドレス・データを，再度アドレス・バス上に乗せアドレスを行なうやり方である。この点の詳細は第7章第9項のアドレス・モードの項目で述べる（図**6-3-2，3**）。

次に，連続的なデータのアクセス，及びヒープ，スタック等のテクニックをサポートするために，アドレス・レジスタ間接には，使用ごとに内容を自動的に更新する機能が附加されている。これは処理データ長によって，１，２，４を加減算する機能である。またアドレスにも加減算が必要だという理由により，アドレスの加減算の命令が用意されている。ただし，アドレスはベース・ページが16ビットの範囲，通常は32ビットの範囲のため，２種の演算範囲しか持たされていない。アドレス・レジスタが扱う語長は，ワードとロング・ワードのみである。

もちろん，このアドレス・レジスタも，インデックス・レジスタになる事ができる。

アドレス・レジスタの特例はＡ７で，これはスタックを使う命令のディフォルト・スタックとなっており，サブルーチン・コール時のＰＣ値保持用のスタックはＡ７が使われる。またスーパーバイザのスタックは，68020／30を除き，割込み＝例外処理時のＭＰＵの内部状態保持のためのスタックとして使われる(68020／30は，割込み専用のスタックＩＳＰを持つ)。このＡ７は２本あり，正確には９本であるが，ソフトウェア・モデルとして直接処理できるのは８本である。２本のＡ７は，ＵＳＰ（ユーザー・スタック・ポインタ）とＳＳＰ(スーパーバイザ・スタック・ポインタ，68020／30ではＭＳＰ：マスタ・スタック・ポインタとＩＳＰ：割込みスタック・ポインタに分かれる）に分かれており，ＳＲの内のＳフラグ（68020／30では更にＭフラグ）によって切替えて使用される。68020／30のＩＳＰは，例外処理が発生しＭＰＵが自己の内部状況をスタック上に展開する時に使われる。まず，このスタックを使い例外処理の一次処理を行ない，その後，スーパーバイザとして複数のタスクが存在しているうちの特定のものに移行しようとする時，それ用のスタック値をＭＳＰにセットし，ＳＲ内のＭフラグを操作する事により，複数のタスクに別個の独立したスタックを持

図6-3-2　メモリ間接アドレス・モードを用いてのアドレス指定

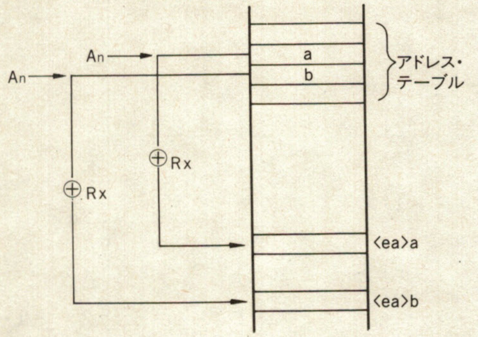
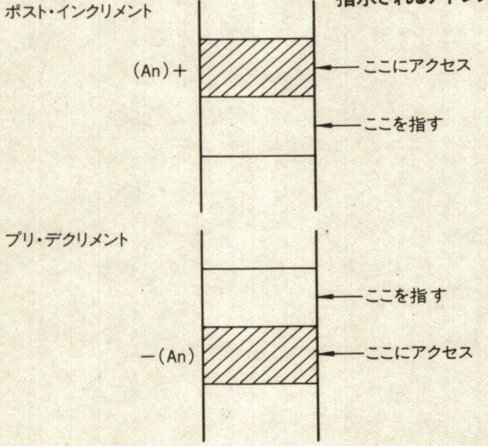


図6-3-3　アドレス・レジスタ関接時の実行前後の指示されるアドレス



6-4　ＳＲについて

ＳＲは16ビットのレジスタだが，下位８ビットはＣＣＲ：コンディション・コード・レジスタと呼ばれユーザーでも，スーパーバイザでも使われる。

ＣＣＲのうち実際に使われているのは下位５ビットで，次の様なＭＰＵが演算，転送を行なった結果の状況を反映している(図**6-3-4**)。

ビット０＝Ｃ：キャリー／ボロー

加減算の時，データを正の２進数表示である（あるいはＢＣＤ演算で）と考えた時に，結果が演算長を超え，桁上げ／桁下げがあったものとして演算された時に１になり，演算結果に桁上げ／桁下げのない時には０になる。またシフト命令，ローテイト命

図6-3-4 68000ファミリのSR内のSフラグとスタック(A7)との関係

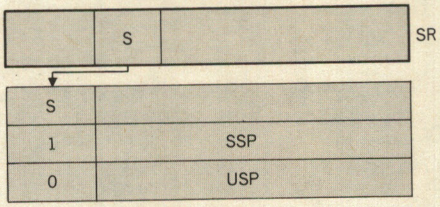
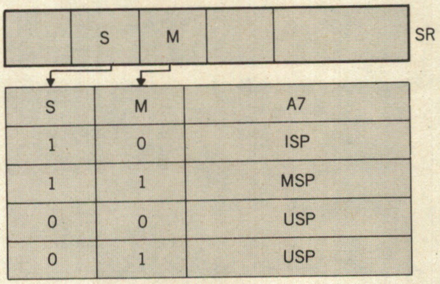


図6-3-5 68020のSR内のM(マスタ)フラグをS(スーパーバイザ)フラグとのA7の関係



68020のA7

図6-3-6 例外処理フロー

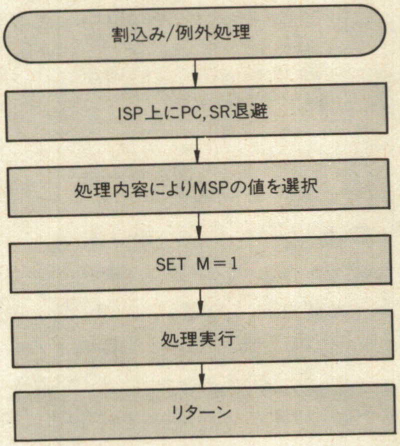
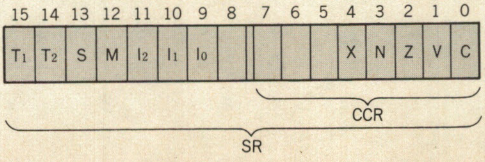


図6-4-1 68020／30のSR



令の結果，最後に移動したビットの内容がここに残っている。

ビット1＝V：オーバーフロー

数値演算を行なう時，そのデータをオフセット・バイナリ（最上位ビットを正負を表すビットとし，全ビット0を数値0として，＋1は最下位ビットのみを1とし，－1は全ビット1という様な表現）としてとらえ，オフセット・バイナリ同志の演算を実行した結果として，結果がオフセット・バイナリの表現範囲を超えた時にセットされる。つまり，8ビットのオフセット・バイナリの表現範囲は－128～＋127であるが，＋100と＋100を加えると，結果は－56となる。この時，このVフラグがセットされて，結果がオーバーフローした事を示している（**図6-4-2**，**図6-4-3**参照）。

ビット2＝Z：ゼロ

演算，シフト／ローテイトの結果，データがその時の演算長で全て0ならば，1になるビットである。

ビット3＝N：ネガティブ（負）

これは演算長の最上位ビットの内容を反映する。オフセット・バイナリ時の最上位ビットは，正負を表すのでこれを表している。

ビット4＝X：エクステンド

これは，数値演算やシフト命令時にCと同様に扱われる。ただし命令によって，これを演算内容に含めたり含めなかったりすることができるため，桁上げ／桁下げの処理にはこれが使われる。

これらのフラグの例は，命令実行時に自動的にセット／クリヤ／ホールドされるが，また命令によってもセット／クリヤすることもできる。

セット／クリヤされたフラグは，特定な命令によって利用され，プログラムのフローを作るのに使わ

図6-4-2 オフセット・バイナリ表現と，その表す値

コード	値
0 1 1 1	+7
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	0
1 1 1 1	−1
1 1 1 0	−2
1 0 0 0	−8

オフセット・バイナリ表現とその表す値

って利用され，プログラムのフローを作るのに使われる。特定の命令は「cc」を持つ命令で，このccとは各種の判断条件を示し，この判断根拠はこれらのフラグである。

ここでコードとは，命令ワードのビット8～11に入る条件指定コードで，これを持つ命令は「Bcc, DBcc, Scc」または特定のフラグのみを利用する命令としては「TRAPV」：Vフラグを利用，「CHK1・CHK2」：Nフラグを利用等がある。更に68020では「TRAPcc, CPDBcc, CPcc」も加わっている（**図6-4-4**）。

6-4-1　ＳＲの部分

ＳＲの部分へのアクセスは特権命令でしかアクセスできない。つまりスーパーバイザ・モードでのみ使われる部分である。

ビット８～10：Ｉ０，Ｉ１，Ｉ２，

これは割込みのマスク・ビットで，割込みの禁止レベルを表わす。リセット時は，全て１にセットされて，レベル７以外の全ての割込みを禁止する（レベル7割込みはNMIで特別扱いとなる。第10章参照）。

ビット12＝M，マスタ／インタラプタ

これは68020のみ持っているビットで，スーパーバイザ時のスタックを例外処理発生時と，各タスクに分かれてからの処理時に切替えるための制御ビットである。

ビット13＝Ｓ：スーパーバイザ

これは，スーパーバイザ／ユーザー状態を指示するのに使われる。

リセット時は，１にセットされてスーパーバイザを示し，ＯＳの働く環境を提示する。この時スタックは，ＩＳＰがＳＳＰとして使われ，プリビレッジ命令も全て使える。このビットのクリヤは，ＡＮＤ命令やMOVE命令で０に戻す，またはＲＴＥ命令でスタック中のＳＲのコピーを戻させる事により実現する。Ｓビットが０になると，スタックはＵＳＰが使われる。プリビレッジの命令は使えず。ＳＲもアクセスできなくなる(**図6-3-5**)。

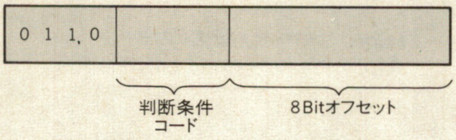
この事に関しては，第４章ＯＳの章を参照されたい。

図6-4-3　オフセット・バイナリの加算例

```
   0 1 1 0          +6
 +) 0 1 0 1       +) +5
 -----------      ------
   1 0 1 1 = -5     +11
```

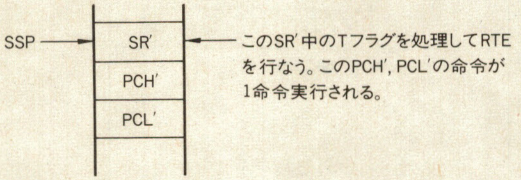
例として＋5と＋6の加算を行なう。単純に10進で加算すると＋11となるが，これは，4ビットのオフセット・バイナリの表現範囲－8～＋7を超えるので，Vフラグがセットされる。Vフラグが立つ事で，ユーザーは，通常の加減算の場合のキャリー／ボロー（桁上げ）の発生を知る。

図6-4-4　条件分岐命令のオペワード



分岐系の命令は，4Bitの判断条件コードにより，PCのとる値を変える。この判断条件コードは，判断条件付，全命令に共通である。

図6-4-5　トレース用スタック・フレーム



トレース命令を有効に実行させるには，PC, SRを同時に回復する必要があるので，スタック中に，1命令実行用のSR', PC'を作って，RTE命令にてこのSR', PC'をMPUにセットして，1命令実行を行なう。このSR'内のTフラグは，もちろんセットされている。

ビット14，15：Ｔ１，Ｔ２：トレース

ビット14は68020／30のみのビットである。68000ファミリでは，ビット15を１にして命令を実行させると１命令実行後，例外処理が起動され，トレースのベクタが参照され１命令実行の結果を評価できる。

68020／30以外では，これはＴビットのみで，単に１命令トレースしかできなかった。68020／30では，Ｔ２とＴ１のコードにより，「BRA, JMP等までトレース」と「１命令トレース」を切り替えられる。

この「BRA, JMP」等の命令とは，プログラム・カウンタで連続的に増加しなくなる所まで実行し，そこで例外処理へと移行する。これには，ＲＴＥやTRAP等の命令も含まれる。

このトレース・ビットは，例外処理が発生すると自動的にリセットされる。

このトレースの実際的な使用法は,「ＯＲＩ」や,「MOVE」命令で,トレースを立てても次の命令はトレースしたい命令にはならないので,必ず「ＲＴＥ」命令を用いて行なう。

例外処理のスタック・フレームをソフトウェアで作成し,その時ＳＲのコピーに対してトレースに当たるビットを立てて,ＰＣに当たる部分は,トレースしようとする命令の入っているアドレスを入れておき「ＲＴＥ」命令を実行する。するとＰＣ,ＳＲの内容がスタックから戻され,ＰＣの示す所から１命令を読み実行して,終わるとトレース例外処理へ移行し,トレース・ベクタの示す所にＰＣ値は移行し,１命令実行した結果のＰＣ,ＳＲはスタックの中に残っている(**図6-4-5**)。

また,「MOVEM」命令を続いて実行することにより全レジスタの内容コピーもスタック中に作る事ができるため１命令実行の結果は完全に評価できる。

またこの時,外部割込みが発生しても,内部割込みは優先権を持っているため,先にトレース・ベクタを取り,スタックにＰＣ,ＳＲを待避させた後に外部割込み処理に入る。外部割込みの処理をしては困る時は,あらかじめ割込みマスク・レベルを７にセットしておく。

例外処理に入れば,自動的にトレース・ビットが０になるので,トレースの処理の中でトレースをする事はない。

68020ファミリ・アプリケーション分類

68020ファミリも,その用途分類がなされる様になってきた。特に68030，68040，また68000ＭＰＵコアのワン・チップ・タイプへとＭＰＵの種類は増えてきているが，これに伴って，それぞれのＭＰＵに向けての応用分野が決められてきている。これを幾つかに分類する。

1)．EWS用

68030，68040となる。これは，ＥＷＳは，ほとんどの場合，仮想マシンでありMMUを必要とする。そして，ハードウェアは，ワン・ボードか，それ程大きくないＶＭＥ等のシステムで，そのＣＰＵはオール・イン・ワン・タイプが多い。そこで，ＭＰＵの直ぐ外側に高速のキャッシュの拡張もよく行なわれる。この条件に適するのは,同期型バスを有する68030，40であり，かつデータ・キャッシュも有効に利用され得る。MMUの能力の限定は，ＥＷＳが，もともと一人で使う為のものであるから問題にならない。

2)．MAP等の通信システムへの応用

68020と68851の組み合わせで，アクセス・レベルの保護を組込むと良い。

3)．マルチ・ユーザー・マシン

これは，多人数で同時に使うシステムで，やはり仮想マシン構成が必要なものである。ものによりデータ・ベースのサポート等も行なう。この場合にも，68020と68851の組合せが勧められる。

これは，多くのタスクを切替えて実行してゆく時に同時に，８つのタスクの変換データをPMMU中に持てる点が重要だからである。

4)．制御用のワン・ボード

このレベルでは,幾つかのＭＰＵが考えられる。高度なものの場合には，68020＋FPCP，中低位なものとしては，68000／68010が良い。もし価格を重点とするならば，セカンド・ソースも多い68000が最右翼となろう。プログラムはＲＯＭ，あるいはＲＡＭにロードして固定なのでMMUは必要としない。

5)．EWSや,UNIXシステムのI／Oプロセッサ

この場合には価格の安い事と，低位のＩ／Ｏディバイスへのサポート信号 (E,VMA) がある事が重要となり，68008か68000が用いられると思われる。このファミリに更に，高い能力の新しいコンセプトのＭＣＵが加わってきてワン・ボード・タイプの場合の分類は更に分かれてゆくであろう。

ともあれ，ＭＰＵの選択は，その内蔵されているハードウェアの内容を良く吟味して行なっていただきたい。又，外付ハードウェアが遅くて，ＭＰＵの能力を生かしきれない事のない様に外付部分を考慮する必要がある。

表１：応用システムから見た68020ファミリMPU　　◎内蔵，○有効，△接続可，—不可

	FPCP	命令キャッシュ	データ・キャッシュ	ＭＭＵの能力	システム保護	数値計算能力	I／Oサポート	仮想マシン
68040	◎	◎	◎	中	中	高	△	◎
68030	○	◎	◎	中	中	中	△	◎
68020＋68851	○	◎	△	高	高	中	△	◎
68020	○	◎	—	—	—	中	△	△
68010	△	—	—	—	—	—	○	△
68000	△	—	—	—	—	—	○	—
68008	△	—	—	—	—	—	○	—

第7章
命令解説

7-1　68000ファミリの命令

ここで説明する命令は，モトローラ社のユーザーズ・マニュアルに記載されているアセンブラ・ベースのものである。したがって，各命令の詳細はアセンブラ解説書か，ユーザーズ・マニュアルをご覧頂きたい。

ここでは各命令の詳細に入る事は避け，特徴ある命令の使い方，またこのファミリの設計思想上の問題を含む点についてのみ，追求したいと考える。したがって，命令の使い方の点では詳解をせず，命令の概略と考え方に主眼を置いて記述する。

7-1-1　データの移動命令

68000ファミリでは，データを移動する時，つまり他のプロセッサの言語なら「LOAD，STORE，PUSH，PULL」等，あるデータのコピーを別の所に作る作業を総称して「MOVE」グループの命令としている。

このアセンブラ書式は，

MOVE 〈ea〉，〈ea〉

であり，左側のオペランド（第1オペランド）から右側のオペランド（第2オペランド）へデータを移動する（アセンブラによっては，オプションでこれを逆にする事ができるものもある）。

データ長は各命令の後に以下の様に指定長をつけて指示する。

MOVE.B（バイト）
MOVE.W（ワード：指定無用）
MOVE.L（ロング・ワード）

この指定長は，他の命令でもデータ長の指定ができるものは同様にして指定を行なう。ただし，命令によっては指定できないものがあるため，その場合には指定は不要である。

このMOVE命令の中で，第2オペランドがアドレス・レジスタの場合は，データ長がワードとロング・ワードに限定されるため，「MOVEA」という別命令にて扱うが，一般のアセンブラは第2オペランドがアドレス・レジスタなら「MOVE」を「MOVEA」に自動的に読み替えるために，MOVE命令と同類とする。

同様に余り意識しないで使う命令に「MOVEQ」がある。これは，イミディエットの1～8の数値をロング・ワードのデータとしてデータ・レジスタに入れる命令である。

また「MOVEI」も同様に第1オペランドにイミディエットの記号のある場合に，自動的に読み替えられる。

一般に，アセンブラはイミディエットのオペランドに対し，命令語長が最も短くなる様に命令の読み替えを行なうので，数値の1～8はMOVEQ，それ以上はMOVEIとするが，もし，デバッグ時等で数値を変えたい時に「MOVEQ」とされては困るのであれば，「MOVEI」と指示すれば良い。こうすると「MOVEQ」への読み替えは起こらない。

〈ea〉について

第1オペランド，第2オペランドに「〈ea〉」と示してあるが，これは「Effetive Address」の略で実行アドレスと訳される。68000ファミリの命令では，命令はその本来の機能のみを表わし，機能の及ぶ所を〈ea〉と表す。「機能の及ぶ」というのは，このMOVE命令で言えば，転送元と転送先の両者であり，その各々に命令に対し，許されているアドレス・モードのいずれかによって示される「実行アドレス」が入る。この実行というのは命令の実行に直接関与するという意味合であり，直接，実行を受けるというのとは別である。具体的には〈ea〉の示す実際のメモリやレジスタを指す。〈ea〉で表わされる68000ファミリの1データの移動は，その長さがバイトであろうとワードであろうとロング・ワードであろうと形式上は同じである。また，アドレス・モードは，ソース（第1オペランド）に関しては無制限で，ディストネーション（第2オペランド：受け側）は，プログラム・カウンタ相対のモードが禁止されているため，PC相対を省く全てのモードとなる。これにより，MPUの内部レジスタ間の転送，MPUのレジスタと外部メモリとの転送，外部のメモリとメモリの転送，スタックへのプッシュ，ポップ，プログラム・ループを構成する命令と組合わせてのブロック転送等がアドレス・モードの選択により行なえる。

7-1-2　他のデータ移動命令

68000ファミリは，32ビットのレジスタが計16本もあるので，レジスタのコピーを作るのではなく，内容を交換したい場合がある。その時は，「ＥＸＧ」命令により，二つのレジスタ間のデータを入れ替える事ができる。

多次元のマトリスク型のデータ・テーブルを扱う時など，一度，あるデータ・エリアのアドレスをポインタに入れ，そこから更にオフセットを取りたい場合があるが，命令の実行ごとに〈ea〉をＭＰＵに計算させるのではなく，数多く使う時には一度計算させてそれをレジスタに保存しておき，次からはその内容を使用，というテクニックを使う方が，実行時間も速く応用性も広く，ループの組みやすい事は容易に理解できると思う。

このための命令が「ＬＥＡ」である。第２オペランドは，ポインタとして使えるアドレス・レジスタでここで指定するアドレス・レジスタ（Ａ０～Ａ７）に第１オペランドの〈ea〉が計算されその値が入る。

これと似た働きをする命令に「ＰＥＡ」がある。この命令は〈ea〉を計算する事は同じだが，その結果を指定したスタックに入れる。この機能を使い，あるサブルーチンに行く時に，サブルーチンを終了してのリターン・アドレスをスタックの中に入れておき，サブルーチンへはジャンプ命令でき，「ＲＴＳ」命令で先にスタックの中に入れておいた〈ea〉の値をプログラム・カウンタに入れ，そこから実行をさせる事ができる。

7-1-3　内部レジスタの退避

割込みや，サブルーチンのプログラムを作る時，68000ファミリのＭＰＵでは，いずれの命令もレジスターレジスタ間の演算が速く，効率が良い。そこで自己の処理に必要なワーク・レジスタを得るためにレジスタをスタック内にプッシュしておき処理の終了時に戻したい時が多くある。特に68000ファミリでは，割込みや例外処理の発生時には，ＰＣとＳＲはシステム・スタック内にコピーが作られるが他のデータ／アドレス・レジスタは保存されないため，必要に応じてシステム・スタックまたは特定のデータ・エリアへ退避してやる必要がある。このための命令が「MOVEM」命令で，オペランドに退避するレジスタ・リストを持ち，このリストにより指定されたアドレスへ内部レジスタを退避する。

このアドレスの指定には２通りの方法がある。特定のアドレスを単に指定し，そこから，数値の大きい方へ展開させる場合と，スタック・オペレーションを行なう場合である(図7-1-1)。

図7-1-1 MOVEM命令の例

```
●Push動作
  MOVEM  A0-A7, $1000
  アドレス・レジスタ全部退避
  MOVEM.L  D0-D7/A0-A7,-(A7)
  全レジスタ待避(スタックへ)
  MOVEM  D0/D1/A0/A1,-(A7)
  D0, D1, A0, A1のみ退避(スタックへ)
●Pull動作
  MOVEM  $1000, A0-A7
  MOVEM.L  (A7)+, D0-D7/A0-A7
```

7-1-4　ローカル・ワークの確保命令

更に，このような割込み内で使われるサブルーチンはメイン・ルーチンでも使われる可能性がある。割込みのある処理で，優先順位によりサブルーチンが使われている時，上位の割込みが入り，同じサブルーチンがその実行中に再度使われる事もあり得る。こういった時に，先の「MOVEM」を使って，空けたレジスタのみで全ての処理が終わっていれば，MOVEM命令が使っている退避用のデータ・エリアが重なっていない限り実行に問題は起きない。このように，多重化して使われても良いように作られているプログラムをリエントラント・プログラムといいマルチ・タスク・マルチユーザーの複雑な実行環境で働くプログラムには重要な手法である。

ところが，いくら68000ファミリのレジスタが多いと言っても，レジスタの数には制限があるため，メモリが必要となる。さて，ワーク用のメモリを得る

にはどうしたら良いか。当然，答えはスタックの中にある。スタックは充分な大きさを与えられているし，この場合，特定のワーク・メモリを要求しないため，スタック・エリアを必要量，自己のワーク・メモリとして確保すれば良い。

こうすれば，同じ処理を何度多重に実行してもワーク・エリアが重なる事はありえないので，多重化した全ての処理が正しく実行される。

このスタック内にワーク・エリアを作る命令が「LINK」，その処理からのリターン直前にワーク・エリヤを解放する命令が「UNLK」である。通常この二つは必ずペアで使われる。**図7-1-2**にこの「LINK」命令の実行の様子を示す。

この命令のポイントはこれが1命令でその実行中に割込みや中断のない事である。

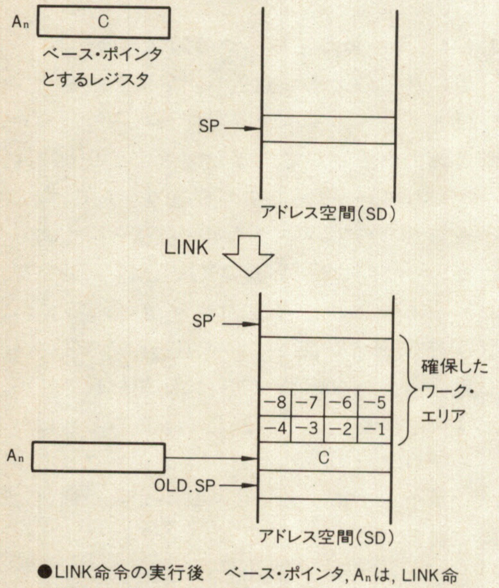
この動作を説明すると，まず指定されたアドレス・レジスタの内容をスタック上にプッシュする。そして，この操作後スタック・ポインタの値をそのアドレス・レジスタにセットする。これでこのアドレス・レジスタは，スタック内のワーク・エリアのベース・ポインタとなった訳である。スタック・ポインタ自体はそれから更に指定されたオフセット分（ワーク・エリヤの量）だけ減らされてワーク・エリアを確保し（リエントラントなローカル・ワークと言う），次の割込み，サブルーチン・コールの使用に供される。このワークは，このサブルーチンが呼ばれるたびにそのサブルーチン内で自分で作るため何回重なって呼ばれてもスタックに割当てられているメモリ量が充分なら問題はない。

また，このワークは図中の記号で判るように，ベース・ポイントから（－1，－2，……）というように負の方向にとられるため，判りにくければポインタをデータのトップに変えても良い。しかし「UNLK」命令を実行する前に，必ず元に戻さなければ移らない。一般にはマイナスのまま「ＥＱＵ」定義分でラベルを定義して用いる。

「UNLK」命令は指定されたアドレス・レジスタの内容をスタック・ポインタにセットし，そこからアドレス・レジスタの待避されていた内容をプルして（このプル動作でスタック・ポインタの値は＋4される）元に戻る。この時に先に確保したローカル・ワークとしてのＲＡＭエリヤは全て無効となるのでCALL元に渡すデータは予め転送しておく必要がある。

図7-1-2　LINK命令の実行前・後

●LINK命令の実行前　Aₙは任意のアドレス・レジスタで内容はセーブされるので，ベースとして使うために，プログラマによって指定される



●LINK命令の実行後　ベース・ポインタ，Aₙは，LINK命令実行前の内容をセーブしているアドレスを示す。そこから，新たなスタック・ポインタの示す点までの間が，確保したワーク・エリアである。

7-2　論理，数値演算命令

一般の論理演算，数値演算は，他のＭＰＵと大差はない。唯一の注意点は，基本的に論理演算，数値演算はデータの演算のため，演算される側，被演算子は，データ・レジスタであるという事である。これは論理演算の一種であるシフト・ローテイト命令群も同様である。

従って，データに対し行なう加算，減算，乗算，除算，比較，補数，論理演算のAND, OR, CLEAR等は第1か第2のオペランドのどちらか，また両方（命令によっては，どちら側にという指定のある場合もあるが）必ずデータ・レジスタが必要とされる。

データ・レジスタの演算長は，バイト，ワード，ロング・ワードであり，乗除算を除き，これら命令は1，2，4バイト長の演算ができる。

ただし，加減算と大小比較はアドレスに対しても必要なため，アドレス演算としてアドレス・レジスタに対して行なえる。またメモリにも直接加減算はできるようになっている。

7-2-1　カウント命令

68000ファミリには，他のMPUファミリによく見られる命令「ＩＮＣ」，「ＤＥＣ」といったカウンタを作るのに便利な命令がないようにみえる。事実，そのようなニーモニックはない。これは，一つには68000ファミリのアドレス・レジスタが，プリ・デクリメント，ポスト・インクリメント機能を持っていて，この機能が，その命令の扱うデータ長によってその加減数を自動的に1，2，4と切替えるからでもある。更に，「ＩＮＣ」，「ＤＥＣ」に替わる命令としてイミディエイトのデータを加減算する場合の特別な命令，クイック・モード加減算命令がある。これが，1～8のイミディエイトのデータを16ビット・ワードの1命令ワード内に組込み，単なる1加算，1減算ではなく，1～8の加減算を1ワード命令で実行できるからでもある(ADDQ, SUBQ)。

7-2-2　乗除算について

68000ファミリは乗除算命令を持っている。これは，データ・レジスタ間で演算される命令で，扱うデータはオフセット・バイナリ，またはストレート・バイナリの数値で，68000／08／10／12では，16ビットと16ビットの乗算，32ビットを16ビットで割る除算，68020／30では，更に32ビットと32ビットの乗算，64ビットを32ビットで割る除算が追加された。

7-2-3　シフト・ローテイト命令

68000ファミリのシフト・ローテイト命令は，データ・レジスタに対して実行される時には，1命令で多数ビットの移動ができる。特に移動回数を別のデータ・レジスタに持たせるダイナミック・シフト，ローテイトは，シフト回数の制限もなく，良く用いられる。シフト回数を命令のオペランドにイミディエイトに指定した時は，1～8回の移動数を定義しなければならないし，メモリを直接シフトする時はデータ長＝ワードで1ビットしか移動できない。

(注)68000／08／10では，ＤＳＰや演算専用ＣＰＵではないので，ハードウェア・シフト機構であるバレル・シフタは持っていない。したがって，ビット・シフトの回数に応じて命令実行のクロック数は増加する。68020／30はバレル・シフタを有するのでシフト回数による実行サイクル数の変化はない。

7-2-4　ビット操作命令

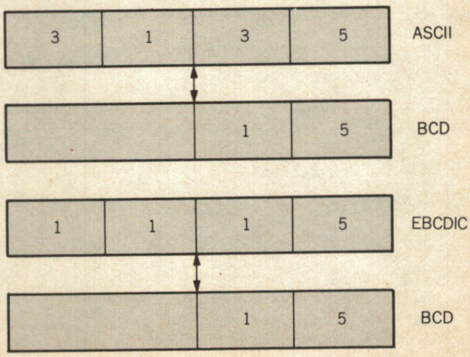
68000ファミリでは，単純な1ビットの操作もできるが，それはセット，クリヤ，反転，テストの4種のみである。また操作する所（第2オペランド）がデータ・レジスタの場合は32ビット全てが操作できるが，メモリに対し直接操作する時には1バイトのメモリしか操作できない。

7-2-5　ＢＣＤ演算命令

10進数サポートとしてのＢＣＤの演算は，68000ファミリでは2桁の演算に限定してサポートされている。2桁の加算，減算，補数計算を行なえる。BCDサポートの機能は，8ビット系のＭＰＵと大差がない。これは，ＢＣＤ数は一部の入出力装置に用いられるのみで，本体内のほとんどのプログラムがバイナリ演算を用いていると考えられているからである。

68020／30では，この他入出力フォーマットのASCIIやEBCDICコードをサポートするため，1バ

図7-2-1　PACK,UNPK命令の動作



イトのＢＣＤ数を２バイトのコード形式に直したり逆に２バイトのＢＣＤ数に直したりする「PACK」，「UNPK」という文字列機能を一部サポートする命令が追加されている(図7-2-1)。

7-2-6　プログラム・フローの制御命令

68000ファミリでは，プログラムの分岐を行なう条件判断分岐命令は，２種の命令に分かれている。一つは単純な分岐を作るために使われ，もう一つは条件判断付ループを作るのに使われる。単純なものが「Bcc」命令群で，ループ用が「DBcc」命令グループである。この「cc」というのは各種判断条件を示し，14種ほどの大小判定，ゼロ／ノン・ゼロ，オーバーフロー等の判断に使う事ができる（図7-2-2参照)。

その他に，無条件指定が「Bcc」，「DBcc」に二つある。

この両者共コンディション・コード・レジスタの内容により，分岐の条件を決定する。また，単なる分岐命令で分岐できる範囲は，ショート・モードで－126～＋128，ロング・モードで－32766～＋32768の範囲への移行，つまり１バイトまたは１ワードのオフセット・バイナリの値をＰＣに加えた範囲（ＰＣ相対）となる。

68020では，更に32ビット・アドレスの全ての範囲に分岐できるようになった。これにより完全なJMP, JSR命令を使わないポジション・インデペンデント・コードを作成する事ができる（エクストラ・ロング＝ロング・ワード・オフセット)。

このデータ長の違いの（分岐のためのオフセット値の）判断は，命令ワードの最初の１ワードの下位８ビットが，ショート・モードの場合のオフセット値の入る位置だが，ロング・モードの場合，ここが「０」となり，エクストラ・ロング・モードの時は「＄ＦＦ」となる。

０も＄ＦＦも，実際のショート・ブランチではあり得ない値である。０の場合，次の命令，＄ＦＦの場合，自分の命令ワードの中間バイト，つまり奇数のアドレスである(図7-2-3)。

(注)「Bcc」系の命令を「ＮＯＰ」の代わりに使う

図7-2-2　Bcc命令とDBcc命令

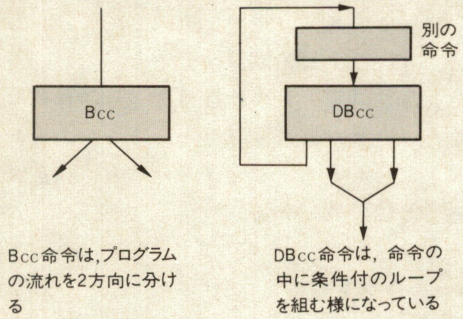
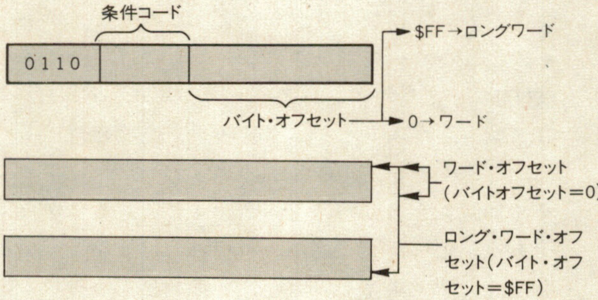


図7-2-3　分岐命令のオペワード



場合，あるいはデバックの都合で「ＮＯＰ」にするつまり次の命令へのブランチは，ショート・ブランチではできず，ロングとエクストラ・ロングで可能となる。

また，このオフセットの計算をする基点は，オペワードの次のワードのアドレスとなっているが，これはオフセットがバイト／ワード／ロング・ワードかに関わりなく一定である。

「DBcc」については，ＭＰＵの種類によらず，オフセットはワードのみである。ループ用の「DBcc」はオペランドとして，データ・レジスタと分岐の行く先を待つ。そして，まず条件をチェックし条件に合致していたら次の命令に進み，ここでは何もしない。

条件が満たされていないと，指定されたデータ・レジスタの下位16ビットを１減算として，－１になったか否かをチェックする。－１になっていたならば次の命令に進み，いまだ－１にならない場合には，第２オペランドのラベルの所にプログラム・カウンタの値が変えられる。

この命令は，68010／12では特にLOOPモード命令を構成する命令として重要な意味を持つ。この命令2ワードと，これに先だつ1ワードの命令を命令デコーダのパイプ中に保存して，ブランチの分岐先をこの先の1ワードとすると，メモリからのインストラクション・フェッチなしでこのループを実行できる。これによって，ブロック転送，ブロック比較，文字，またはデータのサーチ等が高速で実行できる。

さて主題を元に戻し，分岐の範囲の事に触れる。ループ用分岐命令ではアドレス指定法は，ＰＣ相対で同じだが，ショート・モードはない。68000ファミリは，リニア・アドレス方式のため，分岐の範囲が16ビットの境界を越えても問題はないし，オフセット長に関係なく，アドレスの計算は全て32ビットで行なわれている。

7-2-7　単純な分岐とサブ・ルーチン・コール

条件判断を含まない移行命令も使われるが，単なる条件付分岐命令の無条件分岐の場合，「ＢＲＡ」と「ＪＭＰ」の効果はまったく同一である。ただしＢＲＡ命令のオペランドにはオフセット値が入るが，ＪＭＰ命令の場合には〈ea〉が入るので自由度が大きい。この〈ea〉の内にはＰＣ相対も可能なので，この場合のＪＭＰは一種のＢＲＡ命令となる。

同様に「ＢＳＲ」，「ＪＳＲ」もサブルーチン・コールという意味でも同一であり，「ＪＳＲ」の場合，「ＪＭＰ」と同様に〈ea〉としてＰＣ相対も使える（これを使う場合は一般には余り単純な形では使われない。ジャンプ・テーブル中の，いずれかの登録されているアドレスへのサブルーチン・ジャンプのような形で使われる）。

7-3　68020のビット・フィールド命令について

68000では，メモリ用のデータ・バスを通じて外部とやりとりするデータ長は，あくまでバイト単位で１～４バイトの間で行なわれるが，ＭＰＵの命令の上からはこの外部メモリを幅１ビットの細長い空間とみなし，そこに１～32ビットの長さの任意長のデ

図7-3-1　68020のメモリ空間中のバイト・アドレスの振り方

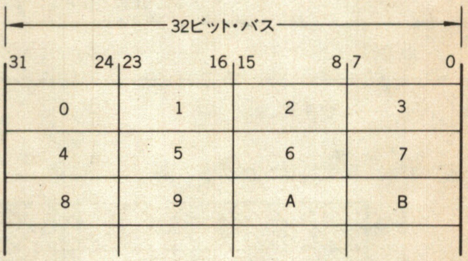


図7-3-2　ビット・フィールド命令のビット番号の振り方

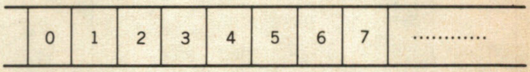


図7-3-3　メモリ空間中のアドレス値

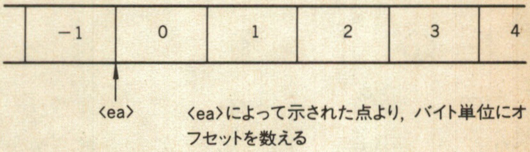
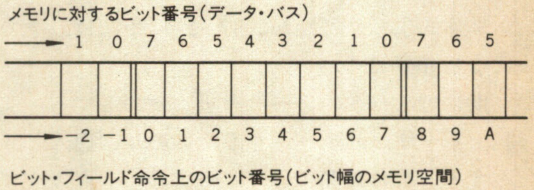


図7-3-4　ビット番号の振り方



ータ・ブロックを並べておき，そのデータ・ブロックを一つの単位として処理すると考える事ができる。

まず，この命令がメモリ空間中に処理すべきデータをどう見い出すかを見る。ここでは判り易いように先の１ビット幅のビット例をモデルとして用いる。

そうすると〈ea〉：実行アドレスで，８ビット＝バイト区切りの特定アドレスがまず指定される（図7-3-1）。

そこの指定されたバイトがベース・ポイントとなり，そこのＭＳＢ＝Ｂｉｔ７を中心に，アドレスの少ない方へ－１，－２，－３……アドレスの大きい方へ１，２，３とビット番号を数える。これをオフセット値とする。このオフセット値は，固定値の場合は０～31まででデータ・レジスタを用いた可変長の場合は，-2^{31}～$+2^{31}-1$のビット長が使える（±４Ｇバイト分。図7-3-2，3，4参照）。

図7-3-5　ビット・フィールド命令の処理部分

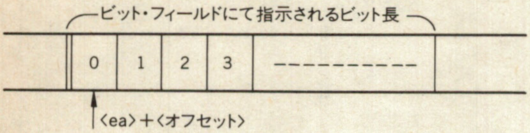
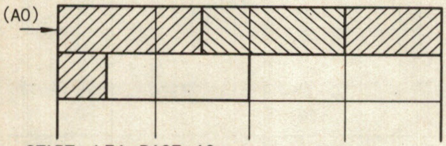
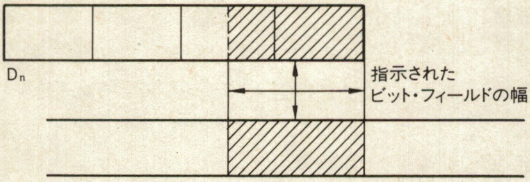


図7-3-6　具体的なプログラム例



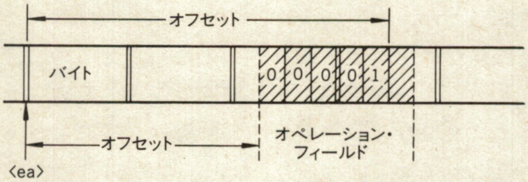

```
START   LEA BASE, A0
        CLR.L D7
        MOVE.L #100, D1
LOOP    MOVE AD, D0 Get Data
        BFINS D0, (A0){D7 : #12}
        ADD.L #12, D7
        DBF D1, LOOP
```

図7-3-7　レジスタとビット・フィールドの関係



ビット・フィールドにて指示されたデータ・ビットは，データ・レジスタの下位部分の対応するビット幅との間で転送される。

図7-3-8　オフセット'がオフセット指示レジスタDnに入る



BFFFO命令は指定されたフィールドの上位よりサーチして，1を見つけると，その時のオフセット値，オフセット'をD_nに入れる

次に，このオフセット値によって指定された特定のビットを基点に，同じ方法で処理されるデータの長さがビット幅として１～31の間で決められる。

具体的に示すと**図7-3-5**となり，手順は以下のようになる(**図7-3-6**)。

仮に12ビットのデータを扱うものとする。そして12ビットずつのデータを順次メモリに記入してゆくとする。この時間のメモリのスタート点は，アドレス・レジスタ：Ａ０によって示されるものとする。

まず，プログラムはこのＡ０に所定のメモリのアドレスをロードし，次にオフセット値のレジスタ：Ｄ７を０にセットする。そして，メモリ中に展開しようとするデータ・ブロック（１ブロック＝12ビットとして）の数をＤ１中に入れておく。

このようにしてデータをラベル「ＡＤ」に代表される所から，データ・レジスタ：Ｄ０へ読込む。更に「BFINS」：ビット・フィールドへの入力命令により，Ｄ０のデータ16ビットの内12ビットをメモリに記入する(**図7-3-7**)。また，基点は12ビットを動かす事はできず，１バイト＝８ビットを動かせばオフセットも４にしなくてはならず，二重手間のため，単にオフセット値にデータのビット長を加えて新たなオフセット値とする（**図7-3-8**)。そして「ＤＢＦ」命令により，データの取得回数が必要値に達するまで（この場合100回），データをメモリ中にすき間なく展開する。

このビット・フィールド系の命令は全部で８種(**表7-3-1**)，フィールドとのデータのやり取りはデータ・レジスタの間としかできない。フィールドのクリヤ，オール１セット，フィールドのテスト，反転等がある。

上記で判る通り，ビット・フィールド系の命令には演算命令はない。演算はデータ・レジスタに移してきて行なうのを基本としている。ただし，このためにデータの移動時にデータをオフセット・バイナリとみて，データの最上位ビットでデータ・レジスタの上位ビットを埋めてしまう命令：「BFEXTS」，単純なバイナリとみて上位を０で埋めてしまう命令「BFEXTU」が用意されている。

まだ特殊なものとしては，フィールド中の最上位ビット（ビット番号０になる）が１の場合のフィールドを探して，その時のオフセット値をデータ・レジスタに入れてくれる命令「BFFFO」がある。

メモリ・ライト時の実際のバス・オペレーションは一度あるデータを読出し，変更を加えてから元へ戻すというリード・モディファイ・ライト動作が行

表7-3-1　ビット・フィールド操作

命　令	オペランドの シンタックス	オペランド・ サイズ	操　作
BFCHG	<ea> {offset:width}	1-32	~ Field → Field
BFCLR	<ea> {offset:width}	1-32	0's → Field
BFEXTS	<ea> {offset:width} ,Dn	1-32	Field → Dn; Sign Extended
BFEXTU	<ea> {offset:width} ,Dn	1-32	Field → Dn; Zero Extended
BFFFO	<ea> {offset:width} ,Dn	1-32	Scan for first bit set in Field; offset → Dn
BFINS	Dn,<ea> {offset:width}	1-32	Dn → Field
BFSET	<ea> {offset:width}	1-32	1's → Field
BFTST	<ea> {offset:width}	1-32	Field MSB → N; ~ (OR of all bits in field) → Z

注：すべてのビット・フィールド操作命令は，指定された操作を実行する前に，BFTST命令の場合と同様にNおよびZのビットをセットする。

なわれる（リード・モディファイ・ライト命令ではない。リード・サイクルとライト・サイクルは，全く別のメモリ・サイクルである）。

7-4　システムの制御に関する命令

システムの制御を行なう命令とは，スーパーバイザ・モードのみで働くか，スーパー・バイザに移行するための命令でＯＳとの関係の深い命令である。

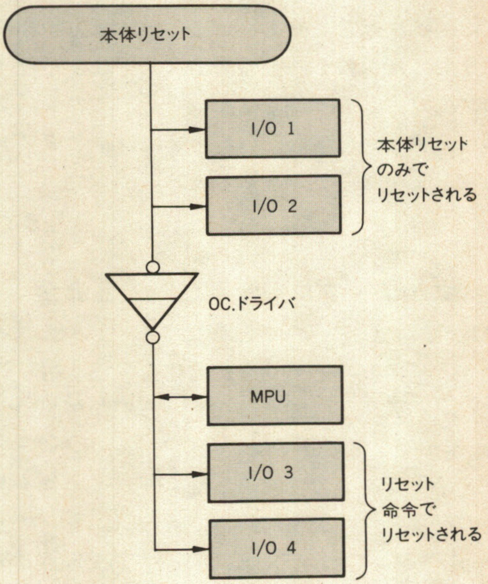
これには，ＳＲに対する読出し，書込み命令の全て，ソフトウェア割込みを発生するための命令（TRAP, CHK, TRAPV,さらに68020ではTRAPcc, CHK2, BKPT），RESET, STOPのハードウェアがらみの命令，また内部のレジスタに対するアクセス命令（MOVE USP，MOVEC），スーパーバイザ・モードでユーザー・モードのメモリをアクセスするための「MOVES」，例外処理からの復帰「ＲＴＥ」がある。

こうした命令の機能はＯＳ本体が使用するものがほとんどで，その機能も逆に単なるＭＰＵのプログラムには不要のものが多い。ＯＳ部分を記述する時にのみ使われるものが主体である。

この中で「RESET」命令は少々変わった命令である。この命令はＭＰＵ自身には何の影響も与えないが，ＭＰＵ自身の電源投入時や，マニュアル・スイッチなどでリセット動作を行なわせるためのリセット・ピンを出力にして，このラインに接がれているディバイスをリセットする。つまり，ソフトで外部の入出力ディバイスに対して回路上のリセットを与える事ができる。

したがって，MPUが働いている上で必要不可欠な部分で，この命令によりリセットを与えてはいけない所は，リセットを２系統にしてＭＰＵからのリセットが入らないようにする必要がある（図7-4-1）。

図7-4-1　リセット命令を有効に活かすリセットの与え方



68020では，プログラムのモジュール化に対応した「CALLM」と「ＲＴＭ」命令を持っている。この命令に関しては詳細はシステムのハード（特に68851 PMMU）によるので後述する。

7-5　TRAP命令

この命令はソフトウェアによる割込みの発生を行ない，０～15まで16種の番号をオペランドとして持つ。

これは普通，スーパーバイザ・コールと呼ばれるＯＳコール，つまりＯＳに何らかのサービス，サポートを要求するための命令とされている。この命令は条件付の命令ではないため，ＭＰＵはこの命令にあうとそのまま例外処理へ移行する。

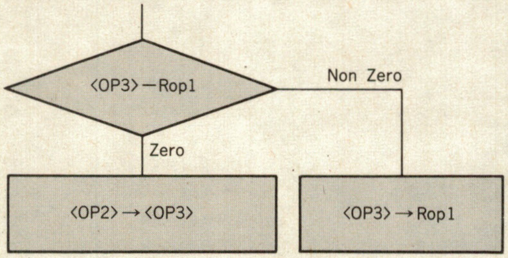
これに対し，「TRAPV」，「ＣＨＫ」，「CHK2」は条件によっては例外処理を起こす命令である。これらは割り算命令で分母がゼロであった時に発生する例外処理と同様，数値演算のためのデータの異常，結果の異常等の修正，チェック，無視を行なうためのものだが，入出力の異常に関係があるため，一度ＯＳに制御を返してやるのが正しい考え方になる。つまり，これらのデータの異常は，それを処理するタスク内で扱えるものではないため，このタスクを起動し，データを与えたＯＳ，または別のタスクに後始末をまかせるという事である。

7-6　マルチ・プロセッサに対する命令

複数のＭＰＵが一つの共通のバス・システムの上で働き，バス上のメモリを共通化して使う事，どのＭＰＵが，アクセス権を持っているのかが問題になった。68000ファミリではこれに対し，リード・モディファイ・ライト命令で，しかもリードとライトの二つのバス・サイクルに渡ってＡＳ（アドレス・ストローブ）を出力したままになる「ＴＡＳ」と呼ばれる命令を持っている。これでセマフォ（互いのフラグの交換による調停部分）を構成していた。

これはある１バイトをテストして，その内容をＣＣＲに反映させ，もし０ならば，該当バイトの最上位ビットに１を立てるという処理をするものであった。これにより，あるデータ・エリアと，あるバイトが結びつけて考えられ，そのデータ・エリアを使おうとするＭＰＵは，対応するバイトにＴＡＳ命令を実行し，もしＺフラグが立っていたら（内容＝０の時）そこのデータ・エリアを使う権利が得られた事になり，Ｚが立っていなければ，そのエリアは使用中という事になる。一度ＴＡＳを使って権利を得ていると，その該当バイトは＄80になっているため他のＭＰＵがＴＡＳを実行してもＺフラグが立たないので使用中である事がそのＭＰＵに伝わる。そのエリアを使い終わったら，該当バイトをクリヤしておかねばならない。クリヤする事により，そのエリアは解放され他の利用を待つ事になる。

図7-6-1　CAS,CAS2命令の演算内容



68020／30では，更にCAS, CAS2命令が追加になった。これらの命令は，三つのオペランドを使用するが，ＴＡＳ命令と同様にバスの使用権を保持したままリードとライトの２メモリ・サイクルを行なうリード・モディファイ・ライト命令である点は同じである（68020／30ではＲＭＣという信号を出力し，ＡＳは各サイクルごとになる)。

この動作は，まず第３オペランドの内容から第１オペランドのデータ・レジスタの内容を引き算してみて，その結果はＣＣＲのみに反映させる。その結果０なら（Ｚ：フラグが立つなら）第２オペランドの内容を第３オペランドに示す所に書き込む。もしＺフラグが立たないなら逆に先に読んだ第３オペランドの内容を第１オペランドのレジスタに入れる。ＣＡＳ２は，この動作を各二つのオペランドに対して行なう。ここで，第３オペランドの示すアドレスは，多くのＭＰＵやタスクに与えるべき資源を管理しているセマフォの一つであり，第２オペランドはその資源を要求するタスク番号やＭＰＵ番号を示している。第１オペランドのレジスタに入る値は，自己が前に使ってもう使っていない，また自己の管理下で中断しても良いタスク番号やＭＰＵ番号を入れこの「ＣＡＳ」命令が行なわれる。すると，もしセマフォが他の優先するＭＰＵに使われていなくて，第１オペランドのレジスタの内容と合致していれば自己の新しいタスク番号かＭＰＵ番号がセマフォに入り，もし使われていれば，どれが使っているかの情報がデータ・レジスタに得られる（図7-6-1）。

7-6-1　マルチ・タスク，マルチCPU用の命令「TAS，CAS」のマルチ・タスクでの応用

これらのリード・モディファイ・ライト命令は，フラグや情報番号などを1命令で，チェック&セットできる。この機能は後述するマルチCPUシステムでは，不可欠のものであるが，もっと一般的には割込み内でのTASKと通常のTASKとの間のフラグやキューのやりとりに使われる。これは，この様なフラグのやりとりは通常のTASKが読んで，次に書く間に割込みが発生して，読んだ内容が割込みTASKの内部で変えられてしまう怖れがあるからで，更に，ある割込みと，それに多重化する別の割込み（タスクのディスパッチのためのリアル・タイム割込みとI／Oのサービスの為の割込み）との間でも，この通信のトラブルは発生し得る。

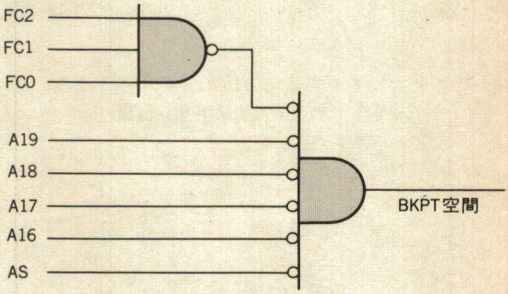
そこで，この様な共用部分の変更やフラグの変更は，間に割込みのかからない状況で行なう必要があるが，スーパーバイザでない状況でも，この様な処置は必要な場合があり得る。しかし，この様な外部からの割込みは，ある1命令の実行途中では発生しないので，このフラグのチェックと変更を1命令で行なえる「TAS，CAS」命令は，どの様なシステムでも大変に重要である。

従来，この様なフラグのやりとりは，共用のメモリ・エリアに，各々一方通行の部分を設定して行なっていたが，これだと通常のプログラム，割込みとも複数存在する時は大変不便なプログラムを組む必要があり，これらリード・モディファイ・ライト命令は，そういった問題をスマートに解決してくれる手段である。

7-7　デバック用の命令

「BKPT」命令はソフトウェアのデバック用の命令で，この命令をサポートするハードウェアと共に用いられ，ある特定の命令でプログラムの実行を中断するか，そのまま実行を続けるかを外付けの回路で決めることができるというものである。また68020／30のみでサポートされている命令である。

図7-7-1　BKPT（ブレーク・ポイント）命令にて使われるBKPT空間のデコード



この命令はオペランドに0～7の数値を要求し，この数値はオペワード内に収納される。この数値はイミディエイトのアドレス・モードで与える

MPUはこの命令を見つけるとCPUスペース（第9章メモリ参照）の専用アドレス・エリアをアクセスし，1ワードの読込みを行なう。

この時のCPU空間をアクセスするアドレスは0～$1Cで，A0，A1は必ず0で，4バイト整合をとってアクセスされる。

そして，オペランドで要求した0～7の数値が，A4，A3，A2に入り，アクセス・アドレスを決定する。

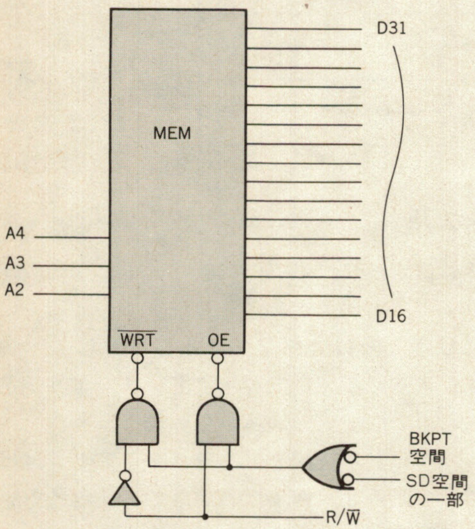
外付ハードウェアは，このCPU空間をデコード（FC2，1，0＝H，A31～A05＝L）する。実際にはFCコードと，CPU空間のタイプを決めるA19～A16をデコードし（図7-7-1），更にBKPT命令の番号と比べ，その番号が使われており，かつその時点ではブレークをさせたくなければ，BKPT命令に代わるオペワードをデータ・バス上の所定の位置に乗せ（一般にA31～A16）アクノレッジを返す（図7-7-2）。

アクノレッジを受けとると，MPUはデータ・バスの上のオペワードをBKPT命令の代わりに実行し，プログラムは継続する。

アクノレッジを返さなければバス・エラーが生じる。バス・エラーが生じると，MPUは不当命令を発見した時に生ずるイリーガル命令の例外処理へと移行し，プログラムにはブレークがなされる（図7-7-3）。

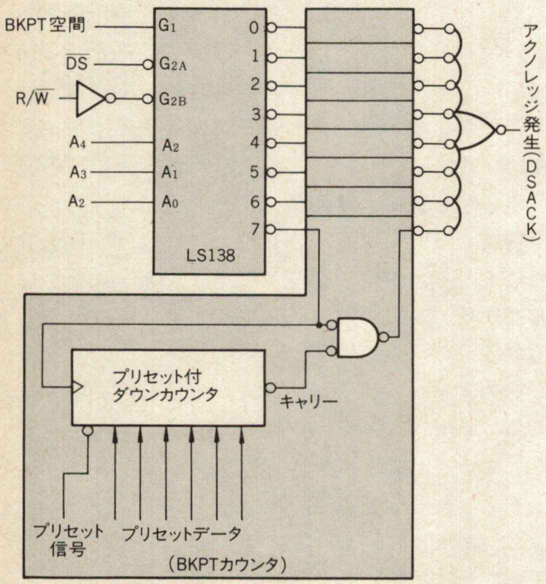
68010／12／20／30では，無条件にブレークするための命令として，イリーガル命令も用意されており，例外処理が起こった後の処理は問題なく作成される。

図7-7-2　BKPT命令実行時にブレークせずに本来の命令をMPUに与えるためのメモリ部分



「SD空間の一部」とあるのはこのメモリの書込み用

図7-7-3　BKPT命令用のアクノレッジ回路



BKPTカウンタが全部で8個あり，各BKPT命令の回数をプリセット・ダウン・カウンタにて数え予めセットした回数が来るとアクノレッジを返さなくなり，ブレークを発生させる

この命令はあるループ・カウント後にブレークしたい時，あるアドレスにアクセスが入った後にこの命令にきたらブレークしたい時に便利である。

具体的手法としてはデバック・プログラムは，ブレーク・ポイントの要求を受け，ブレークしたい命令と条件の指示を受ける。その結果，まずそのオペワードを吸い上げ，そこに何番目かのBKPT命令のオペワードを置く。

次に，外付けの回路のＣＰＵ空間の所定のアドレス（BKPTの１～７の番号に対応している所）に割付けられているレジスタ，またはメモリに吸上げた命令をセットする。そして，条件がループ・カウントならカウンタに値をプリセットし，アドレスならアドレス・コンパレータ用のレジスタに値をセットする。デバック・プログラムを実行してこのBKPT命令をそのまま先の手順でＣＰＵ空間の該当アドレスをアクセスする。

プリセットDOWNカウンタは，まだカウント・ダウンしていないため，この信号はアクノレッジを発生しデータ・バス上に置換えた命令ワードが置かれる。ＭＰＵはこれを読み，正常に実行を続ける。この時このアクセスのネゲイト・エッジでDOWNカウンタを１ダウンさせる。もし，カウントが満了し０になると，アクノレッジの発生が禁じられるので次のアクセスはブレークとなる。

ブレークを発生する条件がアドレス・トリガの場合はプログラムが実行され，何度このBKPT命令を通っても元の命令に，先のカウントDOWN中の要領で置換えられる。ところが，あるアドレスがアドレス比較用のレジスタの内容と一致すると，フリップ・フロップがトリガされ，以後のBKPT命令の該当番号はアクノレッジが返らなくなり，ブレークとなる。

ブレークとなると不当命令例外処理の行く先で，ＭＰＵはソフトウェアにより全アドレス／データ・レジスタの待避，スタック内からのＳＲ，ＰＣのセーブ等を行なって，その内容をコンソールを通じデバッグをしているエンジニアに伝える。

このBKPT命令の特徴は，普通イリーガル命令を使って行なうと，大幅に速度の低下する条件付ブレ

ークをほぼリアルタイムに実行できる事で，MPUの周辺ハードウェアの設計次第で様々な実行上のチェックができるという点である。このハードウェアは(**図7-7-3**)，PMMU68851に内蔵されている。

7-8　68000ファミリのアドレス・モード

68000ファミリの命令はその処理アドレスを「エフェクティブ・アドレス」(＝実行アドレス）により求め，該当部分を対象に処理を行なう。この実行アドレス 〈ea〉は，命令により使えるものが決められている。使える使えないの種別は，各実行アドレスをそのアドレス方式の意味する所で分類し，アドレス・カテゴリ表を作り，このカテゴリにより与えられる。

このカテゴリの表を**表7-8-2**と**表7-8-3**に示す。こ

表7-8-2　実行アドレッシング・モードの分類(68000)

アドレッシング・モード	モード	レジスタ	アドレッシング・カテゴリ				アセンブラのシンタックス
			データ	メモリ	制　御	可　変	
Data Register Direct	000	reg. no.	×	–	–	×	Dn
Address Register Direct	001	reg. no	–	–	–	×	An
Address Register Indirect	010	reg. no.	×	×	×	×	(An)
Address Register Indirect with Postincrement	011	reg. no.	×	×	–	×	(An)+
Address Register Indirect with Predecrement	100	reg. no	×	×	–	×	–(An)
Address Register Indirect with Displacement	101	reg. no.	×	×	×	×	d(An)
Address Register Indirect with Index	110 111	reg. no.	×	×	×	×	d(An, ix)
Adsolute Short	111	000	×	×	×	×	xxx,W
Adsolute Long	111	001	×	×	×	×	xxx,L
Program Counter with Displacement	111	010	×	×	×	–	d(PC)
Program Counter with Index	111	011	×	×	×	–	d(PC, ix)
Immediate	111	100	×	×	–	–	#xxx

表7-8-3　実行アドレッシング・モードの分類(68020)

アドレッシング・モード	モード	レジスタ	データ	メモリ	制　御	可　変	アセンブラのシンタックス
Data Register Direct	000	reg. no.	×	–	–	×	Dn
Address Register Direct	001	reg. no.	–	–	–	×	An
Address Register Indirect	010	reg. no.	×	×	×	×	(An)
Address Register Indirect with Postincrement	011	reg. no.	×	×	–	×	(An)+
Address Register Indirect with Predecrement	100	reg. no.	×	×	–	×	–(An)
Address Register Indirect with Dispiacement	101	reg. no.	×	×	×	×	(d_{16},An)
Address Register Indirect with Index (8Bit Displacement)	110	reg. no.	×	×	×	×	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	110	reg. no.	×	×	×	×	(db.An,Xn)
Memory Indirect Post-Indexed	110	reg. no.	×	×	×	×	([bd,An],Xn,od)
Memory Indirect Pre-Indexed	110	reg. no.	×	×	×	×	([db,An,Xn],od)
Absolute Short	111	000	×	×	×	×	(xxx).W
Absolute Long	111	001	×	×	×	×	(xxx).L
Program Counter Indirect with Displacement	111	010	×	×	×	–	(d_{16},PC)
Program Counter Indirect with Index (8-Bit Displacement)	111	011 011	×	×	×	–	(d_8,PC,Xn)
Program Counter Indirect with Index (Base Displacement)	111	011 011	×	×	×	–	(bd,PC,Xn)
PC Memory Indirect Post-Indexed	111	011	×	×	×	–	([bd,PC],Xn,od)
PC Memory Indirect Pre-Indexed	111	011	×	×	×	–	([bd,PC,Xn],od)
Immediate	111	100	×	×	–	–	#<data>

の〈ea〉は68020／30では種類も増えているため，表も別表になっている。

この表で注意するのは四つに分類された各カテゴリと〈ea〉モードの対比である。例えば，An：アドレス・レジスタの内容はデータではない。ＰＣ相対のアドレス・モードと「#」で表わされるイミディエイト・データは書換えできない＝可変でない等の点である。

このアドレス・カテゴリは次のような意味を持ちもちろん複合して使われる（例：データであり，かつ可変である必要がある。→大部分の演算命令のディスティネーション）。

■データ：このカテゴリに属するアドレス・モードで指定されるものは，データであると考えられる。

■メモリ：このアドレス・モードで指定されるオペランドはアドレス空間中のＲＡＭであると考えられる。

■制御：このアドレス・モードで指定されるものは入出力，他の制御用のアドレスと考えられ，メモリ中の単一のアドレスとなる。

■可変：これは本来的に書換えできるものを指すため，プログラム自身を除いた全部となる。

［注］〈ea〉エフェクティブ・アドレス（実行アドレス）：ＭＰＵは，命令を実行する時には何らかの処理を行なうが，その処理を行なう先を特定しなければならない。様々な処理があるが，その処理内容によって特にアドレス・モードを切替える必要がなく特定のモードしか持たない，または命令自体に含まれてしまうために，特にモードとする必要のないものもある。

このようなものを除き，処理先の指定方法をプログラマが選択，取捨できるものを特にアドレス・モードとしている。そのアドレス・モードによって指定される処理すべき先を，〈ea〉：エフェクティブ・アドレスと言っている。

実際には，「アドレス」といってもデータ・レジスタや，アドレス・レジスタの場合も含まれるので処理の対象先の総称となる。

7-9　各アドレス・モード

7-9-1　データ・レジスタ・ダイレクト

八つのデータ・レジスタ内の一つの内容が処理すべき内容で，データであると考えられる。

7-9-2　アドレス・レジスタ・ダイレクト

八つのアドレス・レジスタの内の一つの内容が処理すべき内容で，それはアドレス値であると考えられる。

7-9-3　アドレス・レジスタ間接

この場合の処理対象となる所は，アドレス・レジスタの内容によって指定されるメモリとなる。68020／30では，この内容全てを使っての指定となり，下位の68000／10では上位８ビットは使われず，下位24ビットが使われる（図7-9-1）。

アドレス値は，68000ファミリでは全てバイト単位に付けられている。68000／10／12／08で，ワードまたはロング・ワードのデータ長のアドレス指定を行なう時は，アドレス・バス・ライン上のＡ０＝０の必要があり違反するとアドレス・エラーとなる。68020ではデータにおいてアドレス・エラーはない。

MPUのタイプによって，外部に出力されているアドレスの範囲は一定ではないが，必ず上位のアドレス内容が無視されてゆく。これによってＭＰＵのタイプにより周辺回路が異なっても，ソフトウェアはそのアドレス使用範囲がイメージにより重ならない限り互換性を保つ。指定先はメモリであるのでアルタラブルでデータともアドレスとも考えられる。

7-9-4　アドレス・レジスタ間接，使用後増加〔アドレス・レジスタ間接，使用前減算〕

この二つのアドレス・モードも，前の単なるアドレス・レジスタ間接と同様，アドレス・レジスタの内容が指示するメモリが処理先となる。ただしこれらは，スタック操作を可能にするような内容の更新機能が与えられている（図7-9-2）。

増加用の方は，アドレス指定にはそのままの内容

を使い，その後データ長相当分，つまりバイトなら1を，ワードなら2を，ロング・ワードでは4を加算する。

減算の方はアドレス・レジスタの内容をまずデータ処理語長分だけ減らし，その後に減らした値の内容でアドレス作業を行なう。

これらの加減算は，命令語の内に含まれている演算長により自動的に選択される。演算長が1，2，

図7-9-1　アドレス・レジスタ間接のアドレス方法

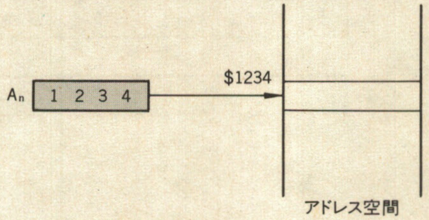


図7-9-2　アドレス・レジスタ間接ポスト・インクリメント付，及びプリ・デクリメント付のアドレス方法とその実行部分

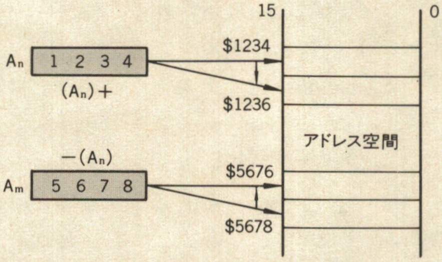
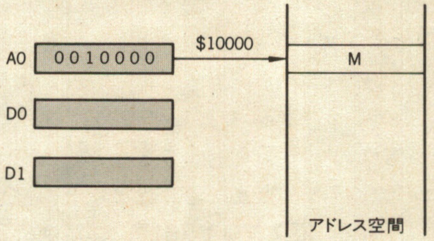


図7-9-3　例A,B,C,Dの実行環境



他　例A.B.C.D.
ポスト・インクリメント，プリ・デクリメントの例
（バイトの場合）

例A　MOVE.B　D0, $(A_n)+$

例B　MOVE.B　D0, $-(A_n)$

例C　MOVE.B　$-(A_n)$, D0

例D　MOVE.B　$(A_n)+$, D0

4バイトでないビット・フィールド系の命令にはこのモードは使えない。

この様子を説明すると，アドレス・レジスタAnの内に＄00010000のデータが入っている。これは間接指定に用いると＄00010000番地のメモリがアクセスされる事になる。ここで例Aの命令を実行するとD0の下位1バイトの内容が＄00010000番地に入りAnの内容は＄00010001になる。

例Bを実行すると，Anの内容は＄0000FFFFとなり，この番地にD0のデータが入る。

例Aの逆操作が例Cである。先に記入したデータを戻すには，Anの内容は＄00010001番地であるべきで，この内容がまず1減算されて＄00010000番地となる。そこのアドレスから，データはメモリの下位1バイトに入り上位は影響されない。

例Dは例Bの逆操作で，Anの内容は＄0000FFFFから始まりこの番地のメモリの内容がデータ・レジスタに入り，その後にAnの内容は＄00010000になる（**図7-9-3，図7-9-4**）。

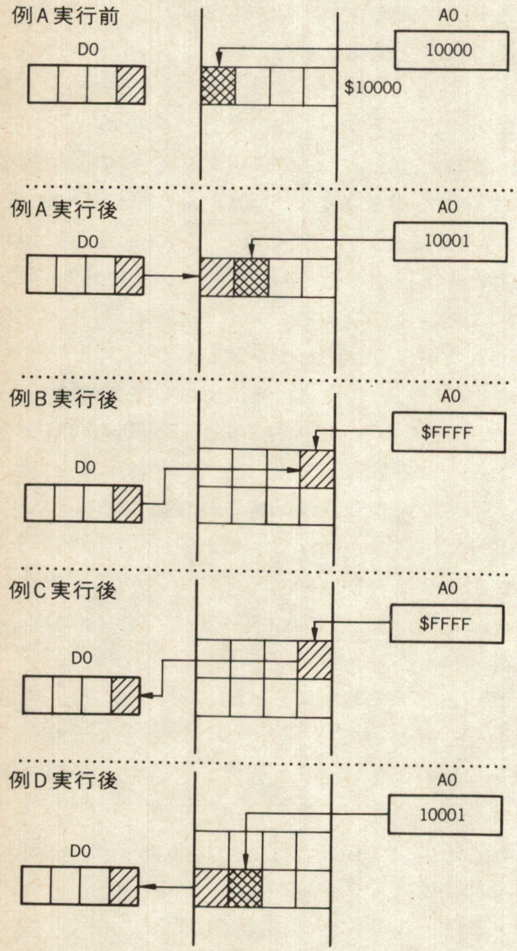
データをメモリ中に連続的に記入し，元に戻すのに少ないアドレスから(An)＋，間接増加モードでアドレス値の大きい方に記入し，−(An)：間接減算モードを使ってデータを元に戻すのをデータ・ヒープ，これと逆に −(An) モードで記入し，(An)＋モードで読出す方をスタックと言う。フリー・ワーク用のメモリ・エリヤの使用法としては，最大アドレス位置にスタックのスタート点を，最小アドレス点をヒープのスタート点とする。

スタックとヒープが交差すると，データの破壊が起きるため，システムは暴走または異常が起きる。スタックとヒープのポインタは交差しないようにプログラム・チェックをかける必要がある。このモードもメモリを扱うのでアルタラブルであり，データである。この良く使われる例がアセンブラのシンボル・テーブルでシンボル・テーブルはヒープ，スタックはシステム・スタックの場合が多い。

7-9-5　アドレス・レジスタ間接，固定オフセット付

これもアドレス・レジスタの内容を元にメモリを指定する。ただし，アドレス・レジスタの内容に命

図7-9-4　実行前後

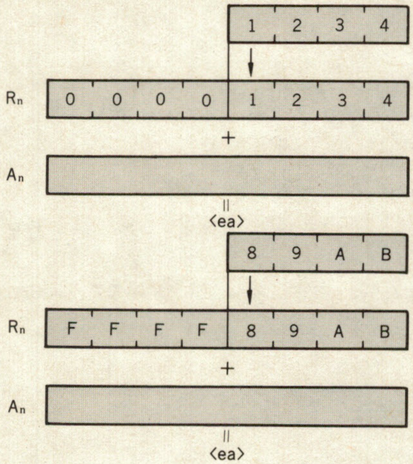


令ワードに続くオペランドに指示される固定値，−32768～＋32767（−＄8000～＋＄7FFF）の値を加算する。この値は，オフセット・バイナリ値として32ビット長で計算されアドレス指定を行なうために使われる。先に同様メモリを扱うので，アルタラブルであり，データである。

7-9-6　アドレス・レジスタ間接，インデックス・レジスタ付

このアドレス・モードは，アドレス・レジスタの内容に8ビットの固定値（−128～＋127）と，もう一つのレジスタの内容を加算して，指定するアドレス値を得る。もう一つのレジスタの値もオフセット・バイナリで扱われるため，≈±2Gバイトの範囲で指定できる。

図7-9-5　オフセット値が符号拡張されて，インデックスとして加算される場合の例



このもう一つのレジスタには，データ／アドレスの全てのレジスタが使える。

テーブル検索によく用いられるのはテーブルの先頭番地はアドレス・レジスタで指定し，テーブル内のポインタ値をインデックス・レジスタに持ち，この値を変えてテーブルのデータを扱う。ベース・ポインタ（この場合のアドレス・レジスタ）と，インデックス・レジスタのみで指定される場合は，固定値(オフセット値)＝0の場合として扱われる。この方法は一般に一次元または二次元のテーブル，またはデータ列の検索に用いられる。

またインデックス・レジスタの語長は，ワード(16ビット)とロング・ワード(32ビット）を使えるが，これは，データ，アドレスどちらのレジスタでも使える。この時はアドレスとして使われるのでデータ・レジスタのバイト使用はできない。また，内部で加算が行なわれる時は，全て32ビットに符号拡張して実行される（図7-9-5参照)。この処理は8ビットの固定長オフセットの方も同じである。

ここで，この符号拡張について注意する事。

オフセットはつい正の値のみで考えやすいが16ビットでは＄＋7FFF～0～＋＄8000の値をとるし，32ビットでも同様の正負の範囲となり，＋側の上限を超えて，負の最大値附近へと落ち込みやすい。

7-9-7　絶対番地指定ベース・ページ

オペランドに指定された値が，そのままアドレスとして処理すべきメモリを指定するのが絶対番地の指定方法で，うち１ワードで指定できる範囲，０～＄7FFFと＄FFFF 8000～＄FFFFFFFFを指定できるのがショート・アドレッシング，一般にベース・ページ・アドレッシングと呼ばれる方法である。

7-9-8　絶対番地指定

命令語に続いて処理すべきアドレスの値を２ワード分，つまり全アドレス範囲を示したのがこの場合である。

このようにオペランドに２ワード・アドレス値，またはデータ値が入る場合，68000ファミリでは必ず上位（ＭＳＢ）側が先に，下位側が後になる。アセンブラ出力を参照すると，オペランドのアドレス値はそのまま命令コードに反映されている。

7-9-9　プログラム・カウンタ相対，固定オフセット付

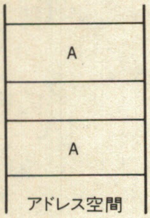
前述のアドレス・レジスタ相対，固定オフセット付と動作は全く変わらない。ただし，アドレス・レジスタの内容はソフトウェアにより任意の値に変化させられるが，プログラム・カウンタ相対では，プログラム・カウンタの値は任意に変えられず，プログラム自体の進行に伴って変わり自由度はない。逆に，ある１命令に限って言えばその時のＰＣの内容は自己の命令ワードのアドレス＋２であるため，一つのプログラムにおいてある１命令と別の命令は，常に同じだけのプログラム・カウンタの差値を持っている。

同じようにプログラム内のデータ・エリアも一定の差値を持っている。この差値をオフセット値として，自己のプログラムと一定の関係にあるデータ・エリヤやプログラム・エリヤを指示するのが，ＰＣ相対アドレス・モードである（**図7-9-7**）。

このモードを用いて，プログラム内を示す全てのラベルを指示させ（一般にアセンブラのオプションで行なう），プログラム自体がメモリ中のどこにいようとも実行できるような組み方をされたプログラムをポジション・インデペンデント・プログラムと呼び，メモリ管理機構を持たないＭＰＵで，マルチ・タスクを行なう時に不可欠なものである（**図7-9-6**）。

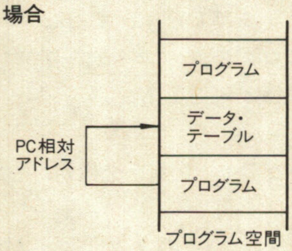
ＯＳ管理下のシステムでは，絶対番地の必要な入出力等はＯＳが管理し，プログラム自体も，ポジション・インデペンデントのフラグがプログラムの性質を示す所にあれば可否をチェックし，可であれば実行用メモリ空間をＯＳの都合によりセットして実行させる。この時のＰＣ値は不明だが，プログラム・フローは全てＰＣ相対で，＋にいくつ，－にいくつ移動（Bcc系命令）と，このＰＣ相対のデータ・アクセスにより可能である。ただし68000ファミリＭＰＵで

図7-9-6　プログラム・モジュールの実行

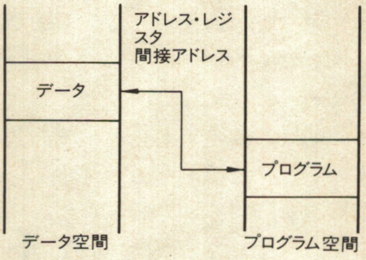


プログラム・モジュールAは，リロケータブル形式で作られれば物理アドレスによらずに実行できる

図7-9-7　ＰＣ相対のアドレス・モードを用いてアクセスする場合



自己のプログラム内のデータをPC相対のアドレス・モードを用いてアクセスする場合の空間とアドレス。この場合は読み出しのみ許される



同一アドレスにあっても，データとして扱うメモリはPC相対アドレス・モードではアクセスしない。アドレス・レジスタ間接を作ってアクセスする

は，自己のプログラム・エリヤが変更され得るディストネーションに，ＰＣ相対を使う事は禁止されているため，FCをデコードしていないＲＡＭ上の，自己プログラム自身，または自己のプログラム内のデータをＰＣ相対で書換える場合は，「ＬＥＡ」命令を用いてあらかじめアドレス・レジスタに該当するエリヤのアドレスを入れておく必要がある。プログラム・エリヤはイメージ的にはＲＯＭなので書換え不能でアルタラブルでもデータでもない。もしMMU等があってプログラム・エリヤを保護しているとしている場合には不可である。

7-9-10　プログラム・カウンタ相対，インデックス・レジスタ付

これも，アドレス・レジスタ間接，インデックス・レジスタ付と同じ動作をする。

ただし前述のＰＣ相対，固定オフセット値付の項で説明した通り，アドレス・レジスタの代わりにプログラム・カウンタが使われる。そこで，命令による書換え側(ディストネーション：第２オペランド)にはこのモードは使えない。

68020／30では，このアドレス・モードの機能が強化されている。それはインデックス・レジスタ値をＰＣに加算する時にそこで処理されるメモリ中データの語長が，バイトなのかワードなのかロング・ワードなのかによって，レジスタの内容の数値がワード数に対応できるようにするため，バイトなら１，ワードなら２，ロング・ワードなら４，伸長ロング・ワードでは８をスケール値としてインデックスに乗算を行ない，その値をオフセット・バイナリ値として加算する。

スケール値は，１(不使用時を含む)，２，４，８の値を取り，レジスタの乗算に使用される。

もう一つ，ＰＣ相対で注意する点は実行アドレスのスペースである。スペースというのはファンクション・コード（ＦＣ０～２）で，スーパーバイザ／ユーザー，プログラム／データ，ＣＰＵと五つのスペースにアドレス空間が別個に存在しているが，このＰＣ相対のアドレス・モードを使った時は，実行アドレスは，アドレス値に関わりなくプログラム・スペースにされる事である。

これは，アドレス・レジスタ間接で指定されるスペースがデータ・スペースのため，メモリ管理機構によって，メモリ・セグメントがはっきりと識別されているシステムでは，ＰＣ相対でアクセスする所は，計算結果のアドレスが同一でもまったく別の所をアクセスする事になるため，コーディング上の注意が必要である。

もちろん，プログラム内にある固定したテーブルのアクセスは，基本的にＰＣ相対のアドレス・モードで行ない，ワーク・エリヤのメモリに存在するテーブルは，アドレス・レジスタによって指示されるのが常識である。このシリーズのＭＰＵのソフトウェアは，当然この常識をもって組まれるべきであるという事が暗黙の了解事項となっており，これに従う事でシステム環境が異なってもプログラムの互換性が保たれる。

更に特殊な例として，プログラム空間内のアドレスを要求したい時がある。この時は，仮想的な値ゼロのＰＣを（ＺＰＣ）用いての相対アドレッシングを行なう。ＺＰＣは値がゼロであるから積極的な意味は持たないが，アクセス空間をプログラム空間に指定できる。

7-9-11　イミディエイト・アドレッシング

これは，命令ワードの直後が処理すべきアドレス〈ea〉となり，そこからデータをとり出して処理するアドレス・モードである。もちろん語長は，バイト／ワード／ロング・ワードとあるが，特殊な例としてはクイック・イミディエイトとして，１～８の値をオペワードに含める事もできる。

このアドレス・モードは当然，ソース側，データを取出す側のみである。また前記の議論で判るように，このデータはプログラム・コード中に含まれるため，この〈ea〉はプログラム・スペースでアクセスされる。もちろん，アルタラブルではない。そしてデータである。

7-9-12　コントロール・レジスタ・モード

これは，ＭＰＵ内のレジスタ，SR, CCR, VBR,

オルタネイト・ファンクション・コード・レジスタ等をアクセスする時に使われるアドレス・モードだが，特殊なため特にアドレス・モードとしては扱わない。コードから言えばイミディエイト・コードの範中となっている。

以上のアドレス・モードは，68000／10／12／08／20／30に共通である。これ以降は68020／30で機能強化拡張されたアドレス・モードについて説明する。

最大の拡張は，二重間接またはメモリ間接と呼ばれるアドレス・モードの追加である。これは，あるメモリ番地を指定し，その指定番地からロング・ワードのデータを読み，読んだデータをアドレス値としてアドレス指定を行なう方法である。

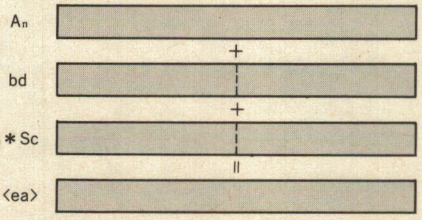
7-9-13　アドレス・レジスタ間接，インデックス，ベース・ポインタ付

このアドレス・モードはアドレス・レジスタ間接の一種だが，16または32ビットの符号拡張されるベース・ディスプレイスメント（Bd）という固定値の一種と，別の16または32ビットのスケール値により1，2，4，8倍され符号拡張されるインデックスレジスタをアドレス・レジスタに加える事によって実行アドレスを決めるアドレス・モードである。

この中でインデックスも，ベース・ディスプレイスメントも，更にアドレス・レジスタ自体もオプションでありどれか一つが欠けていても実行アドレスの計算に問題はない。この系列のアドレス・モードを使う事により変数が三つ許されるので，3次元テーブル・データ配列を扱うことができる。

特にアドレス・レジスタを指定しない場合は，ベース・ポインタとインデックスの値から実行アドレスが計算されるため，両者共データ・レジスタを使った場合，データ・レジスタ間接－インデックス付のアドレス・モードができあがる事になる（図7-9-8）。

図7-9-8　アドレス・レジスタ間接，インデックス，ベース・ディスプレイスメント付アドレスモードの<ea>の計算方法



Sc＝〔1，2，4，8〕のいずれか。

7-9-14　メモリ間接，間接後インデックス付

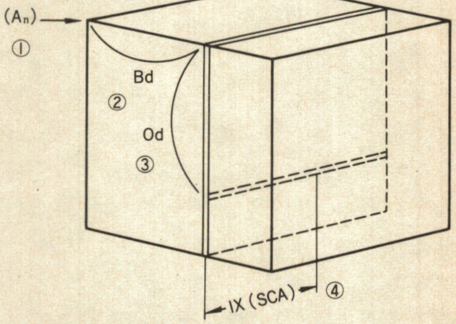
これは68020／30で新たに設けられたアドレス・モードである。この実行アドレスは以下の手順で決定される。

1）アドレス・レジスタとベース・ポインタの値（符号拡張され得る）が加算され，まずメモリ中のアドレス・テーブルの位置を決定する。
2）アドレスされた所から1ロング・ワードのアドレス・データを得る。
3）得たアドレス値にアウター・ディスプレイスメントと呼ばれる別のインデックスの内容を加算する。これはオプションである。
4）インデックス値（符号拡張される）がスケール値により処理されたものを更に加算すると，処理すべき実行アドレスが得られる（図7-9-9）。

7-9-15　メモリ間接，間接前インデックス付

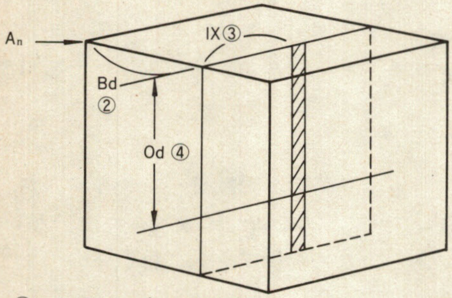
この方式の実行アドレスを得る手順を前項にならって順に説明する。

図7-9-9　メモリ間接・間接後インデックス付のアドレス方法



①にて，データ・ブロックを指定する
②にて，Bd（ベース・ディスプレイスメント）によりあるアドレス平面を指定する
★そこで，この値をメモリ・テーブルより読出す
③そのメモリ平面よりOd（アウタ・ディスプレイスメント）に
④IX（インデックス）値によって交差指定されるアドレスを決定して，データの処理を行なう

図7-9-10 メモリ間接・間接前インデックス付のアドレス方法



①にて，データ・ブロックを指定する
②にて，Bdにより，アドレス平面を指定する
③にて，インデックス作業によりアドレス・テーブルを指定
★アドレス・テーブルよりアドレスを読み出す。
④にて，Odを加算して<ea>とするデータの処理を行なう。

1) アドレス・レジスタにベース・ディスプレイスメントを加算する。
2) 更にインデックス・レジスタの内容が，スケール値倍されたものが加算される（ベース・ディスプレイスメント，インデックスとも符号拡張される)。
3) 2)で得られた値をメモリ中のアドレス・テーブルの値としアドレス作業を行ない，そこから，1ロング・ワードのアドレス値データを得る。
4) 得た値にアウター・ディスプレイスメントを加算する。これも符号拡張され得るし，オプションでもある。
5) 得た結果を実行アドレスとして処理が行なわれる。

これらのメモリ間接のアドレス・テーブルの読出しは，データ・スペースで行なわれるので，ファンクション・コードをデコードしているシステムではプログラム・スペースに置いてあるテーブルの参照に使えない点に於いて注意を要する。この場合にはあらかじめそのテーブルをデータ・エリヤに移すかメモリ管理ユニットによって，共通エリア化しておく必要がある。本来は，プログラム・エリアのテーブルのアクセスは，PC相対を用いるべきである(図7-9-10)。

7-9-16 プログラム・カウンタ相対メモリ間接

これには，前記のメモリ間接の二例のアドレス・レジスタをプログラム・カウンタに置換えた二例がそのまま当てはまる。実行アドレスの計算法は，まったく同じで，アドレス・テーブルがプログラム・スペースにいる必要がある点のみが違う。

7-9-17 インデックス付プログラム・カウンタ間接

このモードで固定値が8ビットの場合は，68020／30以外のプロセッサにもあった。68020／30では，この固定部分がベース・ディスプレイスメントに変わり，16，32ビットが使える。動作は，基本には8ビットの固定長の時と同じで，PC値に符号拡張されて加算され〈ea〉となる。

このモードでアクセスされるのは，FCコードでデコードされるプログラム空間であり，PCがAn（アドレス・レジスタ）であった場合に，データ・エリアがアクセスされるのと異なる。

7-10 アドレス・モードの注意

以上のアドレスモードにより68000ファミリはメモリやI／Oをアクセスし処理を行なう。この時，基本的な構造として以下の各項に注意されたい。

1) メモリに対する<ea>は，モードによりプログラム空間に対するアクセスとデータ空間に対するアクセスが切替わる(FCが変化する)。

PC相対のアクセス（ZPCを含む）はプログラム空間をアクセスする。プログラム空間のアクセスはプログラム読取りの時とこのPC相対の場合，及びイミディエイト・アドレッシングの時のみで，他のメモリ・アクセスは全てデータ空間へのアクセスとなる。プログラム空間にあるメモリは，アルタラブルでないためオペランドのディストネイションに用いて書込む事はできない。

7-11 CALLM命令

7-11-1 モジュール

CALLM命令とこれに対応するRTM命令は，モジュール化プログラム構造をサポートする。このモジュール化プログラムはプログラムの形態の分類の

一方法であるが，タスクやサブルーチンが，機能単位で分類されるのに対し，タスクやサブルーチンが構成するプログラムの大ブロックとしての分類であり，もちろん，機能上の分類ともいえるが，それよりも，プログラム上での優先権上の分類と言えるものである。この68020のＭＰＵシステムで考えられているモジュール化プログラム構造では全体のプログラムは，まず数レベルに分かれる上下の階層構造をとる（図7-11-1)。これはスーパーバイザ（ＯＳ）とユーザーとの分離とは別に行なわれるので，各階層ごとにスーパーバイザの部分とユーザーの部分が存在できる。この階層の数は，１（階層なし)，２，４，８がPMMUによりサポートされる。ソフトウェア上の処理による時は制限はない。そしてこの階層の１階層のみに属するプログラム・パッケージをモジュールとここでは呼んでいる。このモジュールの具体的な意味は，高位の階層（モジュール）は下位の階層のデータ／プログラムの利用が許されているが，下位から上位へのアクセスは許容されない点にある。この関係は，スーパーバイザとユーザーの関係に良く似ている。スーパーバイザはユーザーの空間へのアクセス，ユーザー・モードへの移行が許されているが，ユーザー側からスーパーバイザ側への移行は「TRAP命令，ＣＨＫ命令，TRAPcc命令」等の特定の命令により，割込み機能というＭＰＵ内のハードウェア的な機能を起動する事によりなされる（図7-11-2)。

この様なプログラム自体の上下関係をより細かくサポートしようとするのがモジュール化プログラムである。あるモジュールは，その階層に割当てられた資源の利用権を持ち，この利用権は上位になる程拡大する。これに伴い，上位のモジュールは，システム全体の統合，管理に必要な部分を狙ってゆく事になる。例として，シリアルの入出力の場合を考える。この場合，物理的＝回路的なハンドラ・プログラムは，最下位で，データの受入れとハードウェアのエラーのチェックのみを行なう。ここで得られたデータは次の階層に上がり，システムとしてのエラーの有無，データ要求の有無をチェックする。データ要求があれば，更に上位の階層に，そのデータの

図7-11-1　階層構造プログラムのレベル変移（優先順位の変動）

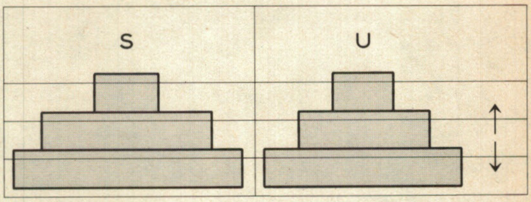


図7-11-2　ＯＳとユーザー間の移行

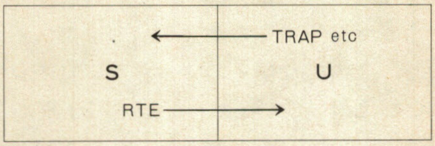
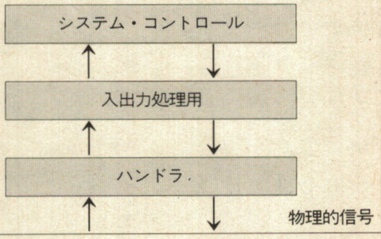


図7-11-3　データの転送手順例



処理を移管する。データが要求される形態にまとまったら，転送先を付して（また要求元を示して）転送を行なってもらうべく更に上位の階層へと制御を渡す。その上位のモジュールは上がってきたデータをデータ要求した所に渡す。その受取り側で，要求したデータの処理が行なわれる（図7-11-3参照)。

この例はあくまでも仮定の例であり，実際の例とは，少々異なる。しかし，概要を示している。ハンドラは，要求元や，データ形式を知る必要はないし知っていても仕方がない。また，どこがデータを要求したかの管理は，そのデータの形式とは関係がない。データの形式は物理ハンドラに近く，要求元の管理はシステム全体の管理に近づく。この様にして全体を管理する部分が上位，部分機能が下位へ割当てられる。更にこの例で判る様に，この場合，各タスク（入出力ハンドラは１タスクでもある）の実行優先権（＝即時にプログラムの実行を要求する優先度）とは別のものである。これは先の例では，最下位のハンドラが実行優先度では，最高位にならねばならないという点からも明白である。

7-11-2　モジュール間の移行

ここで，モジュール構造をとるプログラムがありそしてプログラムの流れが変更されたとする（この時，上から下への流れは，別の命令，例えばジャンプ命令でも可）。この上から下，または下から上への場合により余り意味はないが，同階層へでもプログラムの流れの変更と，モジュールのレベルの＋／－を要求するのが，CALLM命令で，この命令で呼ばれた先から元へ復帰するのがＲＴＭ命令である。

この時，この様な構造のプログラムはモジュール単位にCALLM→CALLM→CALLMと階層を下りまた更に，下から上へ，CALLM→CALLMと上りＲＴＭで下りという様に実行される。この一つのモジュール内では，サブルーチン・コールや，トラップ・コールにより，また様々なプログラム・フローの変化が行なわれ，マルチ・タスクの環境下では，ＯＳ側の制御により，複数のモジュールが，同等に実行される事もあり得る(図7-11-4)。

7-11-3　コール・モジュール命令

この命令はモジュールの階層変化を行なわないとすれば，ディスクリプタと呼ばれるデータ・ブロックを使った間接指定のサブルーチン・コールに等しい。ただし呼ばれる方のルーチンは，サブルーチンとは若干，異なる。それは，呼ばれる側の先頭番地には，受け渡し用のデータ・エリアを示すアドレスをＭＰＵ内のレジスタに渡すために，レジスタ指定のワードが入る事である(図7-11-5)。

先のディスクリプタには，この行き先のレジスタ番号を指定しているワードのアドレスと，レジスタに渡されるデータ・エリアのアドレスが必修である。このコール・モジュールには，いくつかのタイプがあるが，ディスクリプタの先頭には，オプション・フィールド，タイプ・フィールド，アクセス・レベル・フィールドが入れられる（図7-11-6参照）。その次のモジュール・エントリ・ワードには，プログラムの移行先を示し（実際のプログラムは，このエントリの中のアドレス＋２より始まる），データ・エリヤ・ポインタは，エントリ・ワードが指示したワー

図7-11-4　レベル移行の例

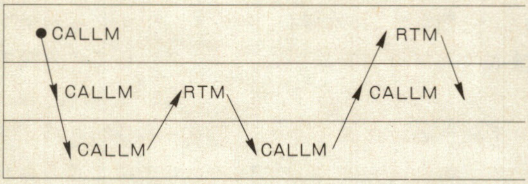


図7-11-5　CALLされるモジュール・プログラムの形式

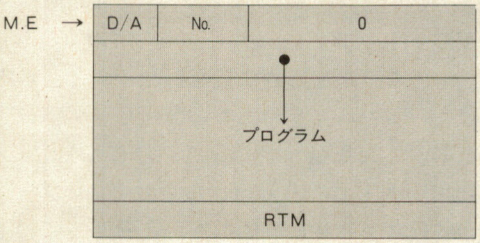
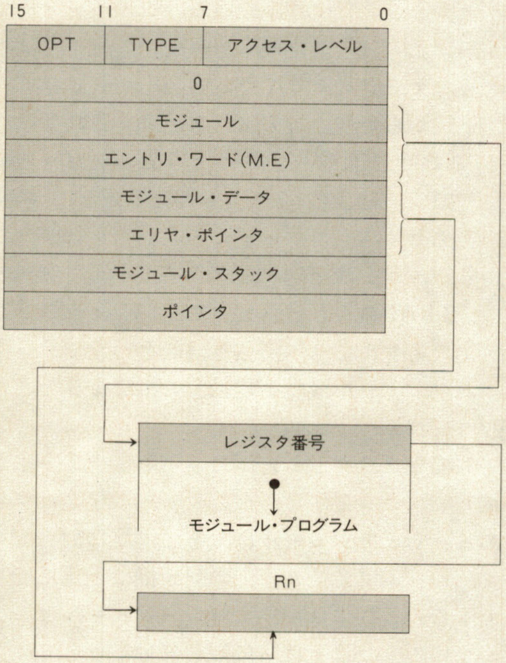


図7-11-6　モジュール・ディスクリプタの構造

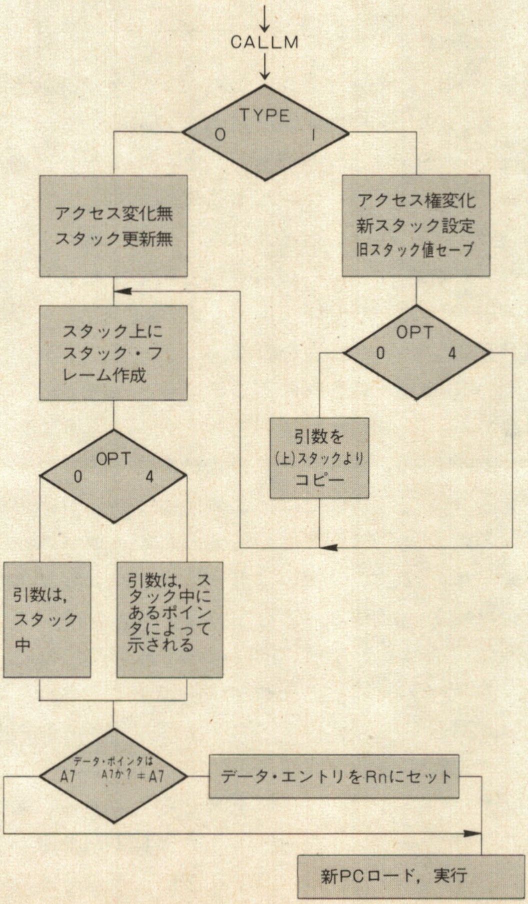


ドにあるレジスタ指定によって指定されたレジスタにコピーされて，データの所在を示す。この時，このレジスタ内の前の内容はCALLM命令によって作られるスタック・フレームという，スタック内のデータ・エリアに保存される。

ディスクリプタには，オプション・フィールドがあり，３ビットが割当てられているが，現在使われ

ているのは1ビットのみで，下位2ビットは常に0でなければならない。従って，このフィールドは，常に，0と4の2値の内の，いずれかをとる。この値が0の時，アーギュメント（引数）と呼ばれるデータ・エリアが現行のスタックの先頭に存在し，そのまま使われるか，もしスタックが新たに設けられるなら，その引数部のバイト数（命令のオペランドで指定されている）分のデータが古いスタックから新しいスタックへ68020によってコピーされ，コールされた側で，このアーギュメントが，スタックの先頭にあった様に見える事になる。オプションの値が4の時は，あのアーギュメントはスタックの中にはなく，アーギュメントの位置を示すポインタが，スタックの内にある事を示す。この場合には，この古いスタックの値が，命令によって作られるスタック・フレームの中にコピーされる。アーギュメント自体はコピーされない。

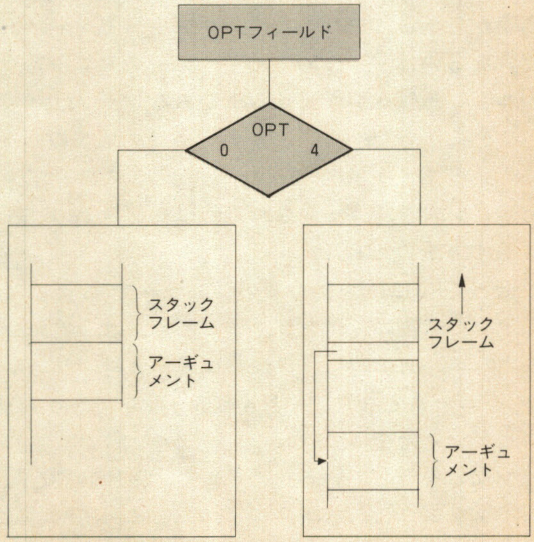
図7-11-7 モジュール・タイプによる命令の形態変化



タイプ・フィールドは5ビットのフィールドだがここも，0と1の2値しか値をとらない。0の時はアクセス権の変化，つまり階層の上下変化はなく，新しいスタックの生成もなく，単に，スタック・フレームを現行のスタックの上に生成する。この時モジュール・ディスクリプタのモジュール・スタックのエントリは使われない。現行のスタックの先頭に新しいモジュールに渡すアーギュメントがあれば，オプション・フィールドは0，ポインタ指定なら4となる（**図7-11-7**）。

タイプ・フィールドが値1をとれば，MPUはPMMUに対し，階層の変化を許すか否かを問い，許されれば，モジュール・ディスクリプタのスタック・フィールドから，新しいスタック値をスタック・ポインタにセットし，オプション・フィールドが0なら，まず，命令のオペランドで示されているバイト数分だけ古いスタックの先頭部分を新しいスタック上へコピーし，その上にスタック・フレームを作成する。オプションが4を示す時は新しいスタックに即，スタタク・フレームを作成する（**図7-11-8**）。

図7-11-8 モジュール・ディスクリプタ内のOPTフィールドによる，アーギュメントの受渡し方法の変化



7-11-4　モジュール・データ・エントリ

これは，スタック中のデータ，またはスタック中のポインタによって示されるデータ以外に，コールする側から，コールされる側にデータを渡したい時にそのデータ・ブロックの先頭番地を命令の実行と同時に，指定したレジスタに渡すものである。コールされる側がこれを受取りたくない時は，レジスタにＡ７（スタック）を指示すれば良い（図7-11-10）。

7-11-5　CALLM命令の概要

この命令は若干，複雑な命令である。それは，一般の命令はその動きの概要は，命令とオペランドで変わるのに対し，ディスクリプタによって動作が変わる事にある。そこでディスクリプタによる形態変化を図7-11-9にまとめる。まず，この命令は，基本的にはサブルーチン・コールである。これは，スーパーバイザとユーザーとの間の変化はなく，単にあるプログラムが別のプログラム・パッケージを呼出しその集約された機能を使うという点では同じである。ただし，タイプ１に関しては，PMMUによりＭＰＵの最上位アドレス３ビットの論理区分によるアクセス制御の変化を受ける。このアクセス制御により，上下の階層を設ける事ができ，この変化を制御して，プログラムの保護，システムの安全性を高める事ができる。これと共に，サブルーチン・コールと異なるのは，二つの方法で，データ・エリアをコールされる側へ引き渡せる事である。一つは，スタック内に置いたデータを渡す方法（またはスタック内のポインタが示すデータ）と，ディスクリプタ内のポインタを，コールされた時にレジスタのどれかに渡す方法である。

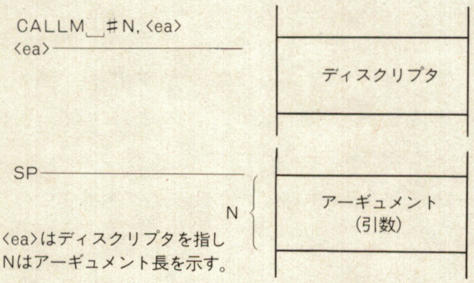
従ってPMMUを持たないシステムでは，タイプ１のアクセス権の変化を要求するときにはこの命令は使えない。その時は，単にパラメータ付のサブルーチン・コールとなり，LINK命令→サブルーチン・コールと大差はない。

MMUを有しているシステムで，階層制御を行なうシステムでは，若干，様子が異なる。各階層は，

図7-11-9 オプションとタイプによるアーギュメント（引数）の関係

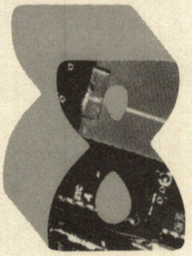
	オプション０	オプション４
タイプ０	引数はスタック中，単にスタック上にスタック・フレームが出来るのみ。スタック値のセーブ無し。	引数はスタック外，単にスタック上にスタック・フレームが出来るのみ。スタック値のセーブ無し。
タイプ１	スタックは新しいスタック階層は変化。引数は新しいスタック中にコピー。	スタックは新しいスタック階層は変化。引数はスタック外，ポインタは旧スタック。引数のコピー無し。

図7-11-10 命令の動き



上位アドレス区分により，論理アドレスが分離されているので，ＯＳ自体が，この管理を持っていなければならないのは当然である。この時，上位の階層から，下位の階層をコールする場合に注意しなればならない。これは，指示されたデータ・エリアのアクセスが許されない事があり得るので，オプション０の引数コピーの機能を使った方が良い。この場合ＭＰＵはブロック転送を行なってくれるので，ソフト側の負担は軽減される。このコール・モジュール命令でもう一つ，サブルーチン・コールと異なるのはコンディション・コードの扱いである。サブルーチン・コールの場合には，この値は保持され，リターンの際，よくフラグとして使われた。コール・モジュールでは，この内容はセーブされ，ＲＴＭ命令で元に戻す時，リストアをされるので，フラグとしての使用法はできない。同時に，帰路に際しては，特定のデータ・エリアを新たに設定することはできない。ＭＰＵ内のデータ・エントリに用いられたレジスタを省く他のレジスタと，データ・エントリのエリヤが帰路にパラメータを渡せる所である。スタック中のアーギュメント・エリアはＲＴＭ命令の実行時に，取り除かれるので，これを使う事はできない。

第8章
68000ファミリのアセンブラ書式



このファミリのアセンブラの書式は当然の事であるが，ソース・コードのみならずオブジェクト自体にファミリとして互換性があり，上位機種に追加されている追加命令を省いては同一である。

各命令の規定書式はユーザース・マニュアルと使用するアセンブラのマニュアルを参照していただく事になるが，以下オリジナルであるモトローラ社のマニュアルによって説明する。

8-1　基本書式

「ラベル␣命令␣オペランド１，オペランド２」

68000ファミリのアセンブラの基本書式は上記の形式をとる。各フィールドの区切りのスペース(␣)は，一つ以上であればよく，フリー・フォーマットをサポートしている。

8-1-1　ラベル

ラベル・フィールドは行の先頭から始まる必要がある。行の始まりがスペースだった場合にその行にはラベルはなしと判断される。ラベル・フィールドの終了はスペース，あるいはコロン(：)で行なう。(：)で終了する時は，先の先頭より始まらなくても良い。

行の先頭にアスタリスク（*）が記入された場合は，その行全体がコメント（単なる文章でコードとは一切無関係）となる。

ラベルは，プログラムのオペランドにより参照されるプログラム・アドレスの位置，メモリ・アドレス値，数値，レジスタ・リストの代替物，代表として使われる。

例外としては，マクロ定義文の時にラベルの位置にマクロ名を置くため，この場合のみ命令エリアで参照される。

8-1-2　命令フィールド

命令フィールドに記述されるのは，アセンブラで定義されている命令のニーモニックが入る。これ以外には，アセンブラに対する指示，マクロの呼出し等が置かれる。68000ファミリでは扱うデータ長が指定できる命令が多いため，この命令フィールドで扱うデータ長を指定する。

命令の扱うデータ長を指定するには，ニーモニックに続いてピリオド（.）を書き，続いてデータ長が指定する文字（B,W,L）を置く。もちろんデータ長の指定できるもので，許されている指定を行なう必要がある。

例えば，アドレス・レジスタの操作に関してはバイト操作は許されないためエラーとなる。

このオプションでデータ長の指示ができる命令で指示を行なわない場合は，特にその命令に指定のない限りワード（.W）が選ばれる。ただしアドレス操作に関係する命令はロング・ワードのものもある。これは，ファミリ全体としては16ビットのデータ長を中心とし，アドレスは32ビット処理を中心としているからである。

このフィールドで分岐系の命令（Bcc，DBcc，BRA）の場合は分岐するアドレスまでの距離を入れてあるオフセット値の演算長の指定が行なえる（S,L, E)。これは，オフセットを計算する際，既出のラベルの場合は最適のフィールド長が計算されて使われるが，未知のラベルが出てきた時は，標準値としてワード長（L：ロング・オフセット＝16ビット）のフィールドがとられる。また，そのラベルがごく近傍あるいは16ビット・オフセットでは届かない遠くにある場合には，プログラマの管理によってプログラムの最適化のためオプションの指定を行なう。

一般にアセンブラは最終結果で，バイト・オフセットで済むオフセット値にワード長のフィールドを与えていた場合，プログラマの注意を促すため警告（ワーニング）を出す。これは，まだ最適化できる事を示しており，エラーではないため実行は正しく行なえると思える場合に出力されてくる。これは，デバックの都合により，時としてロング・ブランチを指定したい場合があり得るので，必ずしも最適化しなければならないものではないからである。

また，68000／08／10／12のオフセット値の演算長は８又は16ビットに限定されているため，エクスト

ラ・ロングの指示はできない（S：8ビット，L：16ビット，E：32ビット）。

8-1-3　オペランド

オペランド・フィールドは命令によって必要とする数が異なる。一つしかオペランドを必要としない命令は第一オペランドのみで，ここに各アドレス・モードの表記の合致するラベル，計算式，ラベルを含む計算式を記入する。

二つのオペランドを要求する命令がこのファミリでは最も多く，一般の演算命令やデータの移動命令等，最も一般的な命令がこの場合にあてはまる。この時，第1オペランド＝ソース・オペランド，第2オペランド＝ディストネイション・オペランド，となっている。

この場合，ソース側はデータの供給元となっており，データは読まれるのみで変化しない。ディストネイション側は，演算または移動を行なわれる先で変化する。

ただCMP，CHK命令ではディストネイション側も変化しない。この場合でも，演算方法は，「D－S」である事は減算命令と同一である。

第1オペランドがデータの供給元とならず，演算事項の指定の場合もある。これは，シフト・ローテイト命令の回数や，ビット操作命令のビット番号等が指定されているような場合である。

オペランドが三つある場合，これはCAS命令等，特殊な命令となる。この場合も，原則に照らして左側がソース，右がディストネイション，中が更新オペランドとなり，右から左へと演算される。

8-1-4　コメント・フィールド

これはオプションのフィールドで，プログラムの内容を文章として記録に残すために使用される。

8-2　アセンブラ中で使える記号

68000ファミリのアセンブラは，まず絶対形式と相対形式に分かれる。

絶対形式は8ビット系のアセンブラで良く使われる方式で，「ORG」ディレクティブによって特定のアドレスが指定され，この指定以後ラベルは全て絶対番地（特定の固定されたアドレス）としてとり扱われる。したがって，そのラベルに対する〈ea〉の計算法も絶対値形式となる。

これに対し，一般に使われるのは，「SECTION」あるいは「RORG」で指定される相対形式で，この形式で指定されているプログラム中のラベルは全てプログラム・カウンタとのオフセット値で表わされるものとされ（一般にPCの値は0からスタートする），これらのプログラムの〈ea〉の計算法はPC相対の計算法となり得る。この形式で作られたオブジェクト・コードは，リンカ・プログラムによって必要な絶対番地を与えられ，幾つも接がれて一つのオブジェクトとなる。この時，アセンブラのオプションで，PC相対のアドレス指定を優先するように作られたプログラムの出力コードは，メモリ空間中のどの番地に置いても動作するため，高級言語のコンパイラは普通このPC相対用のコードを出力してくる。

もちろん，PC相対形式のプログラムでも外部入出力ディバイスや，グローバル変数（システム全体で共用するデータ・エリア）は絶対形式を用い，アセンブラはその場合，絶対形式のラベルに対し，当然，絶対形式のアドレス・モードを採用する。

次に68000ファミリのアセンブラは，命令フィールドで参照される文字列と，オペランド・エリアで参照される文字列をはっきり区別しているため，ラベルがニーモニックと重複しても構わない。しかしオペランドで評価される各レジスタの記号と重なる事は許されない。これを図8-2-1，2に示す。したがって，これらの文字列と同一のラベルは許されない。

もちろんMPUのタイプにより内部に担当するレジスタがなければ許されるがソース・レベルでの上位拡張性に問題が生ずるため使用しない方がよい。

8-2-1　数値および記号

オペランド・フィールドに出てくる（*）記号は現在点でのPCの値を表わす。例えば「BRA.S　*」はこの命令の無限ループを意味する。

数字は先頭に何もなければ10進数，%が付けば2進数，@が付けば8進数，$が付けば16進数を表わす。

数値にはもちろん加減乗除が可能で，その他，シフト，論理和，論理積も使える。

オペランド・フィールドでは，図8-2-3の順に演算していく。計算値は中間結果を含め32ビットの整数で小数以下は切捨てられる。これはあくまでアドレスを演算しているためである。

8-2-2　文字の定義

オペランドの文字は (') アポストロフィでかこまれ，文字列として認識される。文字列中に (') 記号を入れたい時は，この記号を二つ連続して記入する。

8-2-3　定数の定義

定数の定義は「DC」を用いる。このディレクティブは，命令と同様にオプション・フィールドを指定する事ができ，「B, W, L」が使える。ここで指定されたワード長単位に，オペランドに記述された記号，式を定数として，プログラム・コード化する。このオペランド・エリアは通常のオペランドと同様に計算式が使え，結果のみコード化される。文字列を指定した時は，使用している文字数が指定されたワード長に満たない場合で残りエリアには0が詰められる。

8-2-4　変数エリアの定義

変数エリアは「DS」ディレクティブで定義されオプション・フィールドで，バイト／ワード／ロング・ワードの指定を受け，オペランドで，オプションの示すワード長を何ワードとるかを決める。このオペランドは，結果が正の整数値となる計算式のみ，または直接数字のみを受け入れる。

特殊な使い方として，アドレス値を偶数または4の倍数に整合させたい時にこのディレクティブを使う。これは，命令の途中に定数，文字列を挿入した時，バイトのデータに続いてワードまたはロング・ワードのデータ・エリアを定義した時，パフォーマンスを向上させるため(68000／08／10／12ではアドレス・エラー防止のため) 等の時に使う。

図8-2-1　アセンブラ中，及び本書中で使われるN進数指定記号の意味

記　号	
な　し	10　進　法
%	2　進　数
@	8　進　数
$	16　進　数

図8-2-2　アセンブラ中，及び本書中で使われる演算記号の意味

記　号	
+	加　　算
−	減　　算
A／B	除　　算
A＊B	乗　　算
A》N	AをNビット右シフト
B《N	BをNビット左シフト
&	論 理 積
!	論 理 和

図8-2-3　アセンブラ，オペランドにて使われる演算式中の演算優先順位

1	(　)
2	シンボル間の演算
3	シフト
4	論理積・和
5	乗算，除算
6	加減算

図8-2-4　文字の定義例

```
MOVE.L   #'ABCD',DO→「ABCD」が定義される
DC. L    'AB''C'────→「AB'C」が定義される
```

図8-2-5　DS.Wによるワード整合例

```
ERS   DC.B  'ERROR'
      DS.W   0
      LEA   ERS, AO
```

8-3　アドレス・モードのアセンブラ表記

イミディエイト：オペランドに定数を指定する時これは第1オペランドにしか使われない。ポンド・サイン（#）を用いて指定する。

例：ADDQ.L　#$1,D0

　　MOVE. L　#$AD,A0

絶対形式，レジスタ直接：メモリのアドレスからそのメモリ内のデータを指定，またはレジスタの内容を指定する時，この時は直接，レジスタ記号またはメモリに付けられたラベルで表記する。

アドレス・レジスタ間接：アドレス・レジスタ内のデータが示すアドレスのメモリの内容を操作する。この時は，指定のアドレス・レジスタをカッコでかこむ。

例：MOVE D0, (A0)

これのプリ・ディクリメントはカッコの前にマイナス記号を付け，ポスト・インクリメントはカッコの後にプラス記号を付ける。

例：MOVE −(A0), (A1)＋

アドレス，レジスタ間接インデックス付：

インデックス付，固定オフセット付の時は同様にカッコ内に記述する。ただ68020／30以外のアセンブラでは固定値はカッコの前に記述している。

例：

68020／30の場合

MOVE　D０,(OFST，A０，D0)

68000／10／12／08の場合

MOVE　D０,OFST(A０,D０)

68020／30のスケール・バリュー：

68020では，このインデックスの内容にスケール・バリューを乗算でき，この場合は，次の例のようになる。

例：MOVE D1,(A0, D0＊４)

ＰＣ相対の表示：ＰＣ相対の場合，単なるラベルであっても自動的にＰＣ相対のモードが採用される。これは，SECTION等のアセンブラ・ディレクティブによるが，特にＰＣ相対を指定する時はアドレス・レジスタの代わりに「ＰＣ」と表記する。

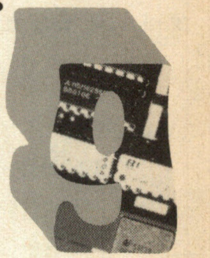
例：MOVE (OFST, PC, D0＊4), D1

メモリ間接の表示：メモリ間接の場合は，そのメモリ間接となる部分を鍵カッコでかこむ。アウター・ディスプレイスメントの付く場合には，外側を更に通常のカッコでかこんで示す。

例：MOVE (〔BOND,A0,D0.B＊4〕), D1

特別な場合，68020ではファンクション・コードのデコードが行なわれていると考えると，プログラム・コードに含まれているデータのアクセスの場合に，プログラム空間へのアクセスが必要だが，ＰＣ相対のアドレス・モードでなければプログラム空間へのアクセスができない。そこで，「ＺＰＣ」が用意されている。これは，値＝０のＰＣとして扱われ，これをＰＣ間接の一種として扱い，プログラム空間のアクセスができる。

例：MOVE　(PDATA, ZPC), D0



第9章
メモリおよびハードウェア

9-1　メモリとメモリ空間

68000ファミリのメモリ空間はバイト単位で表現され，実際に物理的にアクセスされる単位もバイトである。したがって，全ファミリに共通のプログラミング・イメージは，バイト単位のアドレス空間となる。この空間の大きさは，32本のアドレス・ラインで指定できる大きさ，つまり約4Gバイトとなる。

ファミリのバラエティにより，チップの外に出力ピンとしてこの全部のアドレスが出ていない場合には，ソフト上は同一に扱えるが，異なったメモリ空間が重なりあってしまい，多重化されるイメージにより実際に使えるメモリ空間は狭くなる。

68000ファミリでは，このメモリ空間にプログラム自体と，データ，スタック，ヒープ等を置くためのデータ用メモリ，入出力ディバイス，タイマ等を全て割付ける。

ただ，68000ファミリには全て，この正規のアドレス・ライン以外に，ファンクション・コード（FCコード）と呼ばれる3本のアドレス・エクステンションとも言えるラインがある。この出力線はアドレスと同じタイミングで出力され，アドレス線で表現されるアドレス空間の色付けを行なうので一種のページング出力とも言える。

68000ファミリのMMUは，必ずこのFCコードをデコードして，MMUの各ページやセグメントの色付け，分類を行なっている。

ここで表わされるのは**表9-1-1**の通り現在5種が定義されており，一つはユーザーが自由に使って良いアドレス空間となっている。

ここで，UDとはユーザーのデータ用，UPはユーザーのプログラム用のアドレス空間を示しており現在のメモリ・サイクルの内容，つまり専用のページをアクセスしている事になる。そこで，このような分化されたアドレス空間を専用ページ空間と呼ぶ。

同様に，SDはスーパーバイザのデータ用，SPはスーパーバイザのプログラム用のページ空間で，OS用の空間をアクセスしている事を示している。これらの空間は，例えばスーパーバイザのプログラムなら，スーパーバイザ・モードのプログラム・コードの読み取り時にアクセスされるし，スーパーバイザのデータならスーパーバイザ・モード時の命令のオペランドのアクセス時にアクセスされる。

CPU空間は，68000／08では，割込みアクノレッジ・サイクルにしか使われていなかった。

68010／12でも，割込みアクノレッジにしか使わなかったが，スーパーバイザ・モードによりMOVES命令によって，任意の空間にプログラムでアクセスできるようになった。

68020では，CPU空間として，割込みアクノレッジ以外にコ・プロセッサとの通信，BKPT命令（ブレーク・ポイント命令）のサポート，CALLM命令，

図9-1-1　特定の内容を持つ各4Gバイトのアドレス空間

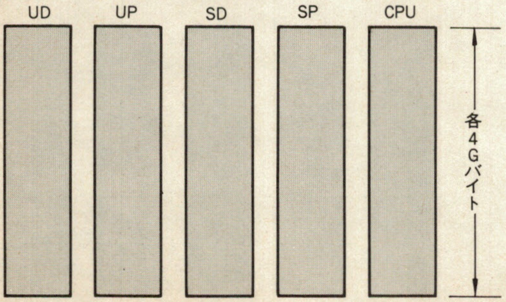


表9-1-1　FCコードによって表わされる68000ファミリの空間の分類

FC2	FC1	FC0	アドレス空間
0	0	0	(Undefined, Reserved)*
0	0	1	User Data Space=UD
0	1	0	User Program Space=UP
0	1	1	(Undefined, Reserved)*
1	0	0	(Undefined, Reserved)*
1	0	1	Supervisor Data Space=SD
1	1	0	Supervisor Program Space=SP
1	1	1	CPU Space

*アドレス空間3はユーザー定義用，0および4は将来モトローラによって使われる予定である。

表9-1-2　各空間に置かれるべきハードウェア，ディバイスの内容

空間	
UD	RAM，ユーザー専用I／O
UP	ROM，（UDとの共用RAM）
SD	RAM，システムI／O
SP	ROM（SDとの共用RAM）
CPU	コ・プロセッサ，RAM，回路

RTM命令のサポート空間として使われるようになった(図9-1-1, 表9-1-2)。

以上の空間の分類を利用するため，FCコードをデコードするシステムでは次の点に注意する。それはソフトウェアにより操作される時に，データとしてアクセスされるメモリ（通常RAM）は全てデータ空間に置かれていなければならず，またROMに入ったプログラムはプログラム空間に置き，RAMでプログラムにもデータにも使う所は，デコードをしないというように，FCコードに対応して接続されるものを区別して使い分けなければならないということである。

各システムで，このFCコードのデコーダは必ず一つは付いている。それは割込みアクノレッジ・サイクル用のデコーダである。そして，それにともなって，I／Oにはこのデコードが与えられている場合も多い。

これを上手に使えばアドレス空間16Mバイトの68000/10でも，ユーザー用のデータRAMを16Mバイト実装し，プログラムやOSのデータ／プログラム，入出力は別の空間に置くという事ができる。

もちろん，小規模なシステムでは，このFCをデコードしなくても良い。その場合には，上記の様な処理区分は不用で，単一のアドレス空間上の処理となる。

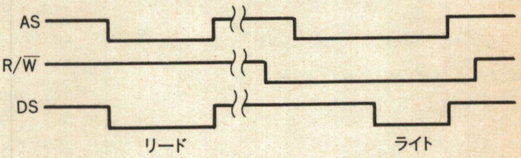
9-2　メモリと命令との関係

9-2-1　命令

68000ファミリでは，プログラムをMPUに読込む機構は，MPUの実行状況からは半ば独立したシステムでパイプラインを構成している。命令解釈機構（インストラクション・デコーダ）と結びついてパイプラインのステージに空ができると，次のプログラム・ワードをメモリ・アクセス機構により読出しては，パイプラインにつめてゆく。したがって，バス上に観測されるプログラム読取りとMPUの実行によるバス・アクセスは必ずしも一致しない。

更に68020／30では内蔵のプログラム・キャッシュ・メモリからプログラムを読む場合があり，この

図9-2-1　メモリ上のデータの変更



メモリ中のデータの変更は，1度MPUに読まれ変更されて再度書き込まれる事によって成される。このRとWのメモリ・サイクルは「リード・モディファイ・ライト命令」を除いては，連続して行なわれるとは限らない。

時は外部バスは使われないので内部の実行とバス上のメモリ・サイクルは完全に離れてしまう。

68000ファミリのアーキテクチャでは，メモリへのアクセスはFC0により明確にプログラムとデータが区別されている。そして，プログラムについては必ず16ビットワードに整合されている必要がある（68020を含むファミリ全てについて，偶数アドレスからの整合を必要とする)。

特に68020／30については，プログラム・メモリが32ビットで構成されている時でも，16ビットごとに命令を詰めておく必要がある。

9-2-2　データ

データの場合は，バイト，ワード，ロング・ワードと，更に68020／30ではハード上は3バイト・アクセスもあり，アドレスはデータ長がバイトであれば全てのアドレスが許される。つまりバイト・バウンダリでアクセスされる。

またデータ長がワード／ロング・ワードの時は，68000／10／12／08では，ワード整合を必要とする(ワード整合：偶数アドレスよりデータを始める事)。ただここに言うバイト長は，ハードウェア上の物理的なバス・サイクルの長さであり，68000ファミリでは各種のリード／モディファイ／ライト・サイクルが行なわれ，下は1ビットから32ビットまで様々な長さのソフトウェア上のデータに対処している。

9-2-3　バス・アクセス

先に，プログラムの読込み機構は実行部と中ば独立していると言ったが，これはバスをアクセスする部分全般にわたっても言える事である。これによっ

て，MPUのレジスタ長が32ビットあり，演算部ではこのレジスタ長に対応した各種の長さの演算を行なう事ができ，そのため各種の長さのデータを要求する事と，実際の物理的なメモリの上にあるデータのアクセスとの整合を得ている。

図9-2-2　68008のアドレス・エラー

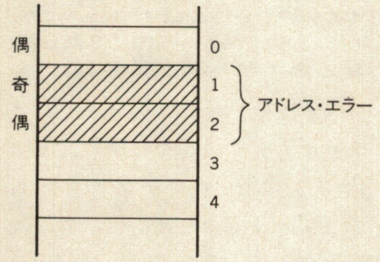


図9-2-3　68000／10／12アドレス・エラー（68020のデータでは可）

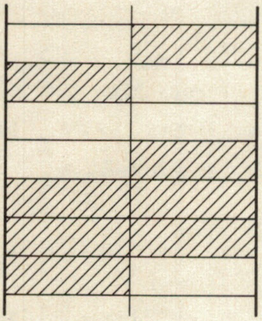
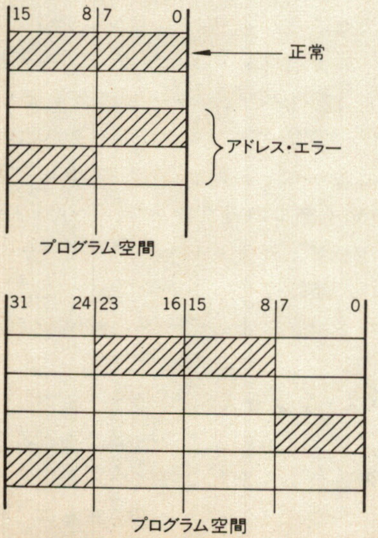


図9-2-4　68020でアドレス・エラーの発生するアクセス（命令ではワード整合が必要）



例えば68020で，ある命令によって32ビットのデータを奇数番地からメモリに記入しようとすると（68Kの普通のファミリではアドレス・エラーという内部割込みを発生する），外部メモリが16ビット幅なら１回目のバス・サイクルでは，第１ワードの下位バイトのみに，２回目のバス・サイクルでは，第２ワード全体に，３回目のバス・サイクルでは第３ワードの上位バイトのみに記入する。この様に１回のデータ転送に多くのメモリ・サイクルがかかる場合と同じく32ビットのデータを４の倍数番地からメモリが32ビットの幅を持っていて，ちょうど32ビットに一致していて１回で書き終わってしまう場合があるが実行部は（つまりソフトウェア上は）メモリのデータ書込みサイクルが何回に分かれようと関係なく単に１データの書込みを実行しているとしか認識しない。

実際には，より多く時間を要するが，これも実行部とは無関係となり，MPUの命令実行部はデータの転送の終了待ちで待ちループに入っている。

要約すると，ソフト上からは68000ファミリの命令コードは，メモリ空間に置かれる時に必ずワード整合を必要とする。データは68020／30では完全にバイト単位であり，他の68000ファミリでは，１バイト長の時はバイト単位でアドレスされ，ワード／ロング・ワード単位の時は，ワード整合を必要とするという事になる。

9-3　回路上から見たメモリ

メモリに対するMPUの要求は，MPUのタイプによって３種に分けられる。

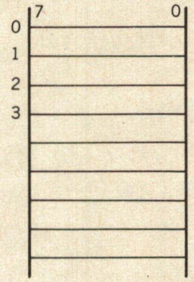
３種の区別はMPUのデータ・バスの幅が１バイトか，２バイトか，４バイトかによりなされる。

【８ビット】68008

データ・バスの幅が１バイトのメモリ空間は最も簡単である。単にＡ０～A19をアドレスとし，Ｄ０～Ｄ７のバイト幅のメモリを構成し，LDS/UDSの区別もない（**図9-3-1**）。

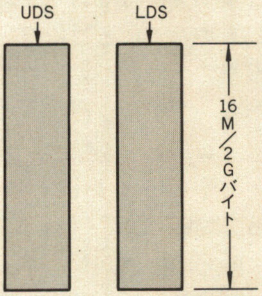
ＤＳ（データ・ストローブ）とR/Wによりタイミングを選びDTACKをMPUに返せば充分で，従来

図9-3-1　68008のバス



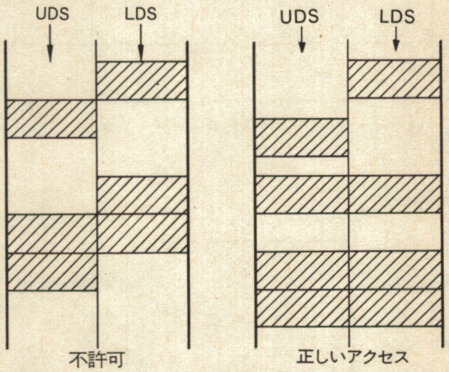
68008のデータ・バスは8ビット幅である

図9-3-2　68000／10，68012のアドレス空間



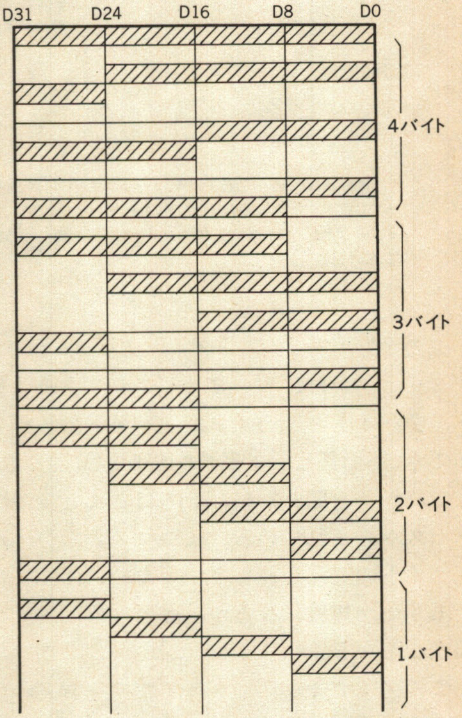
68000/10においては8ビット幅で16Mバイトの空間2つを合成。68012では8ビット幅で2Gバイトの空間2つを合成。各空間は偶数アドレス，奇数アドレス空間である

図9-3-3　68000／10／12のアクセス



68000/10/12では，ワードの境界を超える様なアクセスは許されない。LDS, UDSの片側アクセスは，バイト・アクセス等のみ許される

図9-3-4　68020のデータのアクセスとして許されるアクセス形式



の８ビット系のＭＰＵとほぼ同じである。

〔16ビット〕　68000／10／12

68000ファミリの代表格である16ビット・データ・バス・ファミリのメモリの構成方法には特徴がある。それは二つの８ビット幅のメモリ空間に，LDS/UDSを各々にストローブとして与えて，全体として16ビット幅のメモリ空間を形作る(**図9-3-2**)。

こうしておく事により，ＭＰＵが１バイトをアクセスした時は，LDS/UDSのどちらか一方が(偶数番地ならＵＤＳ，奇数番地ならＬＤＳ）出力され，２バイトまたは４バイトのアクセス時には両方が同時に出力されてデータ長の可変に対応できる。しかしこのシリーズには２バイトと４バイトのアクセス時には，偶数番地からのアクセスでなければならないという規則がある。つまり，**図9-3-3**のような形のアクセス（ワード整合のずれた場合）はアドレス・エラーの例外処理を引き起こして，実際に行なう事はできない（データでもプログラムでも)。したがって，ワード（２バイト）またはロング・ワード（４バイト）のアクセス時，そのアクセスが奇数番地でない事をプログラマが保証しなければならない。

〔32ビット〕　68020／30

68000ファミリの最上位機種は，命令やアドレス・モードの拡張だけでなく，バスの形態においても格段に進化している。

図9-4-1　68000ファミリでのメモリ空間の考え方（バイト×アドレス空間）

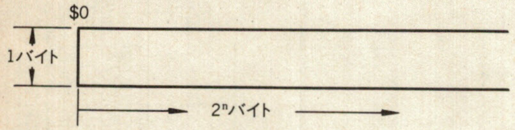
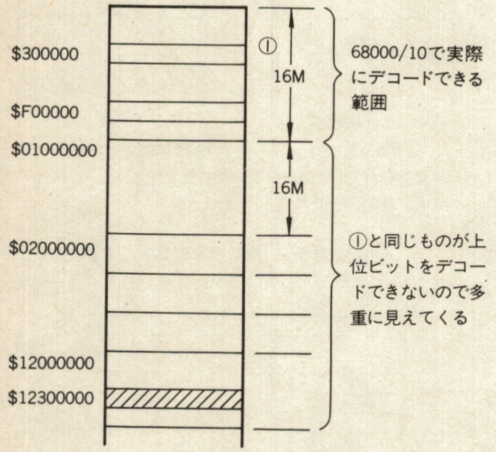


図9-4-2　メモリの重なり（イメージ）



68000/10では，内部では4Gバイトのアドレス範囲を持っていても，上位1バイト分が外に出力されていないので，イメージとなってしまい上位をアクセスしても同じものを見る事になる

このテクニックは，「ダイナミック・バス・サイジング」と言われ，データ・バスのアクセスを１バイト，２バイト，３バイト，４バイトの４種をサポートでき，かつ16ビット系の制限が取り除かれている。

この様子を**図9-3-4**に示す。ほとんど全ての形のアクセスが許されている。またバスの幅自体が１，２，４バイトの混在を許しているため，回路的には若干メモリ側が複雑になるが，融通性は高まり無駄な空間がなくなる。

このダイナミック・バス・サイジングの詳細は，第９章12項を参照されたい。

9-4　MPUから見たメモリ

68000シリーズのMPUは全て，メモリ・マップド・I／Oを採用しており，MPU自体にはMMUやセグメント・コントローラは付加されていない。したがって，MPUから見たメモリ空間は常に単一の１バイト幅で，特別な区切りのないものとして見える。

そのメモリ空間の大きさはどのファミリでも，４Gバイト＝2^{32}バイト分となる。ただ実際のMPUはパッケージやチップ面積上の制限により，68008の２Mバイトから68020の４Gバイトまで何種かに分かれる。

このアドレス空間を縮小されているものでも，MPUの内部アドレスは32ビット分持っており，内部的には32ビットとして動作もできる。ところが，外部に上位アドレスの出力が出ていないため，ちょうど外部で上位アドレスのデコードを行なわない時に出るメモリの重なり（イメージ）という型で，デコードされていない所が見えてくる（**図9-4-1**）。

例：68000（16Mバイト）に於いて，

68000にはA24～A31がない。

由に，＄00300000＝＄12300000

のように，上位２桁分は全てイメージとして重なって見える。そこで，これを下記のように表わして，MPUの種別に対応する場合がある。また，これはMPUの外部デコードを省略したシステムにおいても有効である。

例　68000では　＄ＸＸ１２３４５６
　　68008では　＄ＸＸＸ１２３４５

9-5　68000ファミリのアドレス表記法

この点に関してもファミリのデータ・バスの幅により差が出る。特に68020のアドレス表記は回路構成にかなり依存する。

〔８ビット〕　68008

８ビットの場合は８ビットのMPUと同じで単純に＄０から$FFFFFまで増加していき，８ビット系の場合０から$FFFFまでの間に境目がなかったのと同じように特別の区切れ目などはない。

ただしベース・ページは存在し－＄8000～＋＄７FFFまでであるから，実際には０～＄７ＦＦＦと＄F8000～＄ＦＦＦＦＦの２区間がベース・ページ

となる。内部レジスタとメモリの関係を**図9-5-1**に示す。

〔注〕：この場合でもベクタ・テーブル，命令，ワード／ロング・ワードのデータ等は偶数番地からの整合を必要とする。

〔16ビット〕　68000／10／12

この場合には，ＵＤＳで指定される列の８ビット幅メモリ空間側を全て偶数，ＬＤＳで指定される列を全て奇数とする。そして，ワード／ロング・ワード長メモリの指示は先頭の偶数番地側で行なう。

この場合，内部のレジスタとメモリ間のデータ転送が行なわれる関係は，ワードの時はメモリに付けられたビット番号とレジスタのビット番号がちょうど対応する。

ＵＤＳはUpper Digit Strobe, LDSはLower Digit Strobeの略であり，このUpperはメモリに付したビット番号の上位，Lowerはビット番号の下位を意味する。

ロング・ワードの場合は**図9-5-2**のように内部のレジスタとメモリの写像になる。

〔32ビット〕　68020／30

32ビットのＭＰＵはダイナミック・バス・サイズイングのために，あるメモリの部分が何バイトの幅を持っているかによって数え方が変わってしまう。幅が１バイトならそこではアドレスは一つずつ，２バイトなら二つずつ，４バイトなら四つずつと各メモリが対応してアドレスが付けられていく。ただしこれに関わるのがＡ０，Ａ１，Ａ２のためメモリのデコーダに特殊な細工をしない限り，メモリに対するアドレスの振り付けは２バイト幅のメモリは偶数整合を，４バイト幅のメモリは４の倍数整合を一般に必要とする(**図9-5-3**，**図9-5-4**)。

そして，もう一つ重要な事は，68020に於いてはＭＰＵのデータ・バスの32ビット幅の内，メモリの幅を１バイトとした時はビット番号の大きい(上位の)方から１バイト分，つまりＤ31～Ｄ24が使われる事である。同じように２バイト幅のメモリのときはＤ31～Ｄ16までが使われる。

内部レジスタとメモリの転送の関係は，68020／30ではワード整合の問題がないため指定したアドレスから順に上位のバイトから対応がとられる(**図9-5-6**)。

この時，１バイト幅（Ｄ31～Ｄ24）なら上位から順に，もちろん指定するアドレスは偶数／奇数に関係なく実行できるし，２バイト幅なら(Ｄ31～Ｄ16)**図9-5-6**のように２通りの対応が可能である。４バイトのデータ幅には，１バイトずつずれて４通りの方法が可能となる(**図9-5-7**)。

図9-5-1　68008の場合のデータ・レジスタとメモリ中のデータをやりとりした場合の対応関係

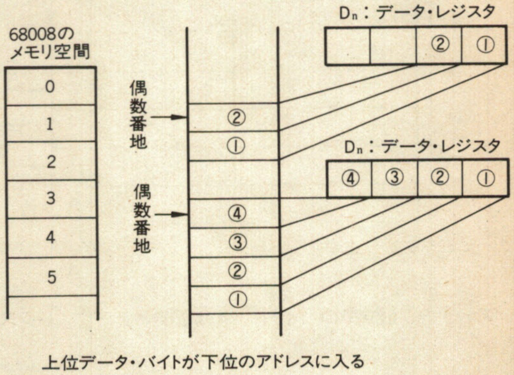


図9-5-2　68000／10／12の場合のデータ・レジスタとメモリ中のデータの対応関係

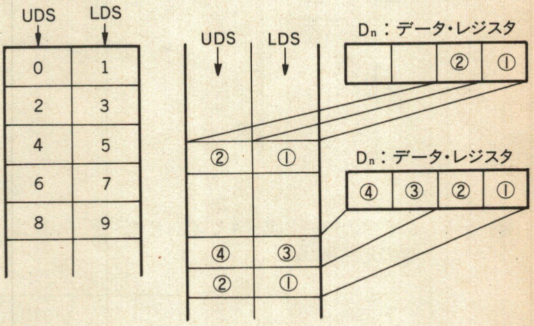


図9-5-3　68020のメモリ空間構成上の物理的整合

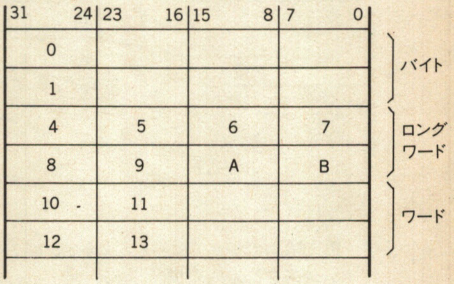


図9-5-4　68020のメモリ空間上，不可能な物理アドレス配置

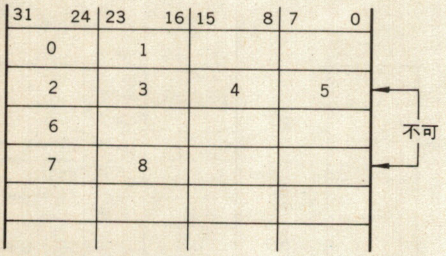


図9-5-5　68020のバイト幅メモリ空間とデータ・レジスタの対応関係

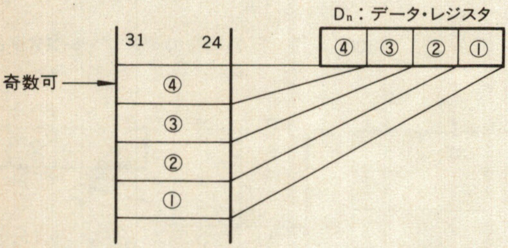


図9-5-6　68020のワード幅メモリ空間とデータ・レジスタの対応関係

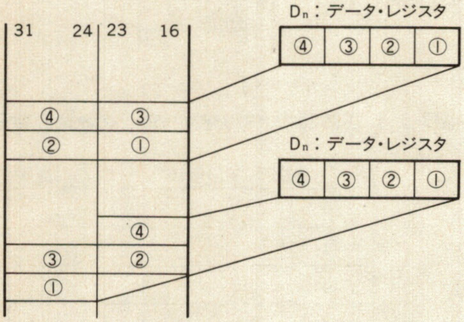
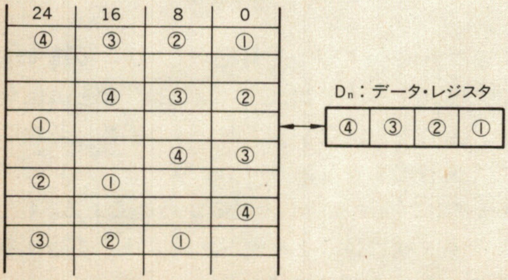


図9-5-7　68020のロング・ワード・メモリ空間とデータ・レジスタとの対応関係



もちろん4バイト幅のメモリに1バイトの転送，2バイトの転送もできる。この時は，内部レジスタの下位バイト側から使われる。また，この関係はビット・フィールド系の命令を用いて任意長のデータを使った時も同じである。

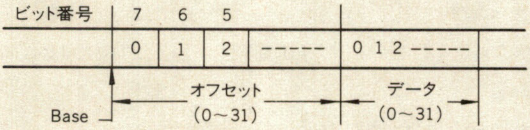
9-6　ビット・フィールドとメモリ

68020／30のビット・フィールド命令では独特のメモリ・ビットの数え方をする。まず，ビット・フィールド命令は，ある特定のバイトのアドレスをベース・アドレスとして必要とする。そこから，当該バイトのビット7を0とし，ビット6を1とし，以下同様に，ビット0が7，次のバイトのビット7は8と最大31までオフセット値を与え，そこで決まったビットを今度はデータ・フィールドの最上位ビットでビット番号を0として，ビット・フィールドのビット長を決め，与えられた命令のオペランドを決定する。もちろん，このフィールド長も32ビット以下でビット番号は31までである(**図9-6-1**)。

9-7　パイプ(インストラクション・パイプ)

命令の先読み機構と実行部を接ぐFIFO（ファースト・イン・ファースト・アウト）のレジスタをパイプと称している。動作は通常のシフト・レジスタと同じだが，実行された命令が，プログラム・カウンタの値を変更した時は全ての内容が捨てられる。命令の先読み機構と合わせて，1命令の実行に要するサイクル数＝時間が，その命令に必要なメモリ・アクセスに使われるサイクル数より多ければ，1命令の実行中にバス・サイクルの空き時間ができるた

図9-6-1　ビット・フィールドによるメモリ中のデータの指定方法



ビット・フィールドではバイトのメモリを横に並べて，1ビット幅のアドレス空間を想定して扱う

め，これを利用して，次の命令またはオペランドをパイプ内に読込んでおき，MPUの実行効率の上昇を図るための機構である。

特に，イミディエイトのアドレス・モードにおいては，このイミディエイト・データがこのパイプ内に存在し，即利用できるため効率が著しく良くなる。

また68010／12では，このパイプが3段あり，ここにDBcc命令を2ワードと1ワードの命令を収納した場合，ループ・モードと称し，新たな命令コードの読取りをせずに2命令のループを実行できる。

これは，一般にいうブロック転送命令や，ブロック比較命令に相当する動作を68010／12にさせる事か可能となっている。

このパイプによって，MPUは幾分かは各命令を並行に処理する事になるが，**図9-7-3**で命令Aの内容によっては，命令Bが実行されない事もあり得るため，一応，各命令は独立を保ち遂時実行となっている。

悪い点としては，時としてこの命令Bの読取りにメモリの応答待ち時間のためのアイドル・サイクルが必要な場合，そのサイクルが2重に効き実行時間が計算値以上に伸びる事がある点である。

更に，この命令パイプは各命令の予備チェックも行なう。つまり，次に実行される命令コードが読み取られると，パイプの最上位では，命令コードの正当性をチェックし，無効命令コードの検出を行ない該当すると例外処理を発生させる(68020／30ではその命令が実際にデコーダに入る時点で発生)。したがって，プログラムの最後の命令がBRAやRTSで次の命令エリアに不当コードが入っていたりするとこれを検出する事はあり得るため，命令コード・エリアは余裕をとることが必要である（最後にNOPを二つ入れると良い)。

9-8　命令先読み機構
（インストラクション・プリフェッチ）

この機構は前述の命令パイプと共同して働き，命令パイプに空きができると，つまり実行部でパイプの内容が使われてしまうと，パイプに新たな命令と思われるメモリをアクセスしてその内容をパイプに入れてゆく。その手順は，あくまで現在のプログラム・カウンタの値を元に，＋2，＋4，＋6したアドレスの内容を順に読出しては，パイプに入れるという手順をとる。

この先読みの命令のアクセス時に，バス・エラーが発生すると，MPUはそこのアドレスが現在の実行と無関係であってもバス・エラーの処理を起こすため，バス・エラーの解析時にはそれが命令フェッチだった場合には，実行アドレスとの間にずれを持っているので注意が必要である。

図9-7-1　68000ファミリの命令読込み機構とインストラクション・パイプの関係

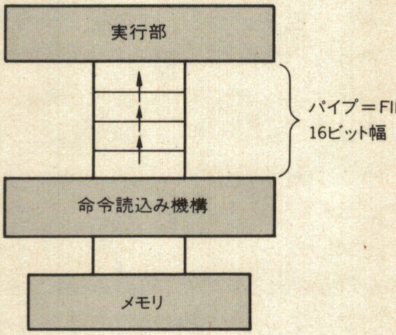


図9-7-2　イミディエイト命令時の実行部とイミディエイト・データの関係

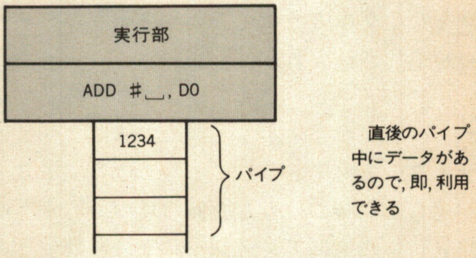
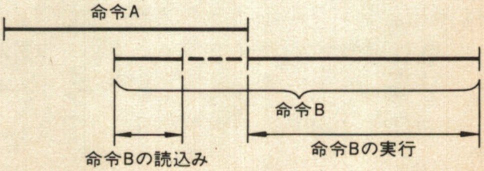


図9-7-3　命令先読み機構による各命令の実行の重なり



またこのため，プログラムをメモリの終端まで詰めこむと，思わぬバス・エラーを起こす事がある。

9-9 メモリのタイミング

68000ファミリのMPUは，メモリのアクセスをストローブに対してアクノレッジを返すというハンド・シェークで行なう。また各ステップは，アクセス・ステートに分けて定められ，この1ステートはMPUクロックの各半サイクルからなる。68020／30では，このステートで6ステート（3MPU・クロック・サイクル），68020／30以外では8ステート(4MPU・クロック・サイクル）が最低必要となる。

メモリのレスポンス・スピードが充分でない場合はアクノレッジを返すタイミングを遅らせる事によりMPUのクロックに同期して遅らせる事ができる。

メモリのレスポンスに充分な時間がとれたらアクノレッジを返す。これは68020／30では，現在アクセスしているアドレスに接がれているバスの幅が，1，2，4の，どのバイト数であるかを示すDSACK0とDSACK1の組合わせによって，アクノレッジを返す。他の68000ファミリでは，DTACKを返す。

MPUはアクノレッジを検出すると，リードの時は1クロック後にデータを取入れ，その後の半サイクルでメモリ・サイクルを終了させる。

更に68030以上では，メモリ・サイクルを2クロック・サイクルで行なうクロック同期のバス・サイクルが追加されている

一般のメモリ・サイクルはこれで終了だが，何らかのエラーが発生すると別の方法により，メモリ・サイクルを終了させることができる。アクノレッジが返らない時に，もしも何の処理も取らない場合はMPUは永遠にアクノレッジを待ち続ける。この時，アドレス，FCコード，その他も出力されたままになるため，MPUの状態を静的レベルで観測する事ができる。システムのハードウェアのデバッグに応用していただきたい。メモリ・サイクルの異常終了はバス・エラー信号を与えることで行なう。その場合，MPUに2種類の対応を要求できる。

1） バス・エラーのみを与える場合

この時は，バス・エラーのみを与えるが，68000/08ではDTACK以前に68010/12/20／30では，DTACK/DSACK以前または以後½クロック以内に与える。これにより，バス・エラーの例外処理を起こせる。

バス・エラーの入力により，MPUの内部処理は様々だが，その点は例外処理の章のバス・エラーの項を参照されたい(第10章3，4項)。

2） バス・エラーとHALTを同時に与え，先にバス・エラーを，次いでHALTを解除した場合

エラーの起きたメモリ・サイクルを始めから再実行する(リトライ・サイクル)。

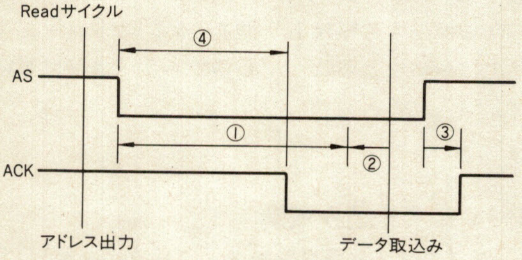
9-9-1 メモリ・システムの設計上の注意点

メモリ・システムの設計は，68000ファミリでは特に困難な所はない。少々難かしいのは，メモリのアクセス・タイムの余裕がどれ位あるのか，逆に何ナノ・秒のメモリはどの位のウェイト・サイクルを予定すべきなのか，が重要である。この計算法は，68000ファミリでは一様であるが以下に紹介する。

〔リード・サイクル〕

アクセス・スタートは，DSかAS（またはLDS，UDS）から測る。リード・サイクルではこのAS, DS, UDS, LDSのタイミングは同じであり，ここではDSを中心に話をする。ライト・サイクル時はDS,LDS,UDSは著しく遅れるので共用する場合はASが良い。データ・シートによりクロックから

図9-9-1 68000ファミリのメモリ読取りタイミング



①はメモリのアクセス時間
②はMPUのデータ・セット・アップ時間
③はACKのネゲイト時間
④はASよりACK迄の時間でメモリ側にて設定される。

どれだけ遅延されて出力されてくるかを確認しておく。68020／30では，このＤＳが出力されてくるのはＳ１，他のファミリではＳ２，そこからの遅れ時間を**表9-9-1**に示す。

次に，アクノレッジのチェックを始めるのは，68020／30ではＳ３，他のファミリではＳ５である。アクノレッジ信号は，このステートに先だってＭＰＵに与えられている必要がある。このセット・アップ・タイムを**表9-9-2**に示す。

１クロック後に，ＭＰＵにバス上のデータを取入れて良いと許可を与えるタイミングとして，あるクロック数の遅延を与える時には，このデータ・セット・アップを計算に入れる必要がある(**表9-9-4**)。

もし，遅延サイクルなしでＭＰＵにデータを読ませようとすると，メモリ・アクセスのＤＳ(UDS, LDS）から，アクノレッジまでの時間の許容値は，68020／30で「１クロックの時間―ＤＳの遅延―ＡＣＫのセット・アップ・タイム」となる。

他のファミリでは，「1.5クロックの時間―ＤＳの遅延―ＡＣＫのセット・アップ」となる。

ここで，データのセット・アップとＡＣＫのセット・アップはほぼ等しく，**表9-9-2**のＤＳ→ＡＣＫの時間に各１クロック・サイクルの時間を加えると，ＤＳからメモリ・データまでの許容値が計算できる(**表9-9-4**)。

メモリのアクセスに待ちサイクルを入れると，入れたクロック数×クロックの時間幅の分だけ量子化されて遅くなっていく。ちなみに，１ウェイトを入れた場合を，**表9-9-5**に示す。

この時間関係は，あくまでＭＰＵのピンに対応しているもので，各データ・バス・バッファによる遅延，アドレス・デコードによる遅延は含まれていない。

特に，68000ファミリはアドレス・ラインの数が多く，多数のラインをデコードしなければならない。そこで，デコーダは数10nsecの時間がかかるのは普通で，特に68020／30では更にＳＩＺコードの解析もメモリ・システム内で必要なためで，より多くの時間を費やす。そこで，ウェイト・サイクルなしのメモリを要求すると，デコーダの遅れを考慮に入れると，おおむね**表9-9-6**の位のメモリをＭＰＵの近傍に配置

表9-9-1 タイミング信号のディレイ(nsec)

MPU	AS, DSのディレイ
68030-25	18
68030-16	30
68020-33	15
68020-25	18
68020-16	30
68000-16	60
68000-12	40
68000-8	40

表9-9-2 各タイミング入力のセット・アップ

MPU	ACK	DATA
68030-25	2	2
68030-16	5	5
68020-33	5	5
68020-25	5	5
68020-16	5	5
68000-16	10	7
68000-12	10	10
68000-8	10	10

表9-9-3 ノーウェイトの為のACK許容時間(nsec)

MPU	DATA to ACK(MAX)
68030-25	(40)×1.0－18－2＝20
68030-16	(60)×1.0－30－5＝25
68020-33	(30)×1.0－15－5＝10
68020-25	(40)×1.0－18－5＝17
68020-16	(60)×1.0－30－5＝25
68000-16	(60)×1.5－60－10＝20
68000-12	(80)×1.5－40－10＝70
68000-8	(125)×1.5－40－10＝137.5

表9-9-4 ノーウェイトの為のデータ許容アクセス時間

MPU	DS to DATA(MAX)
68030-25	60
68030-16	85
68020-33	40
68020-25	57
68020-16	85
68000-16	83
68000-12	150
68000-8	262.5

表9-9-5 1ウェイト時のデータ許容アクセス時間(nsec)

MPU	アクセス許容時間
68030-25	100
68030-16	145
68020-33	70
68020-25	97
68020-16	145
68000-16	143
68000-12	230
68000-8	387.5

表9-9-6 デコーダを考慮したノーウェイトの為のデータ・アクセス許容時間(nsec)

MPU	アクセス許容時間
68030-25	40
68030-16	65
68020-33	20
68020-25	35
68020-16	60
68000-16	60
68000-12	120
68000-8	220

図9-9-2 68000ファミリのライト・サイクル・タイミング

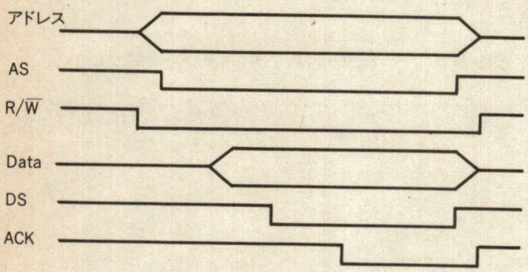


図9-9-3 68020/30のRMCサイクル

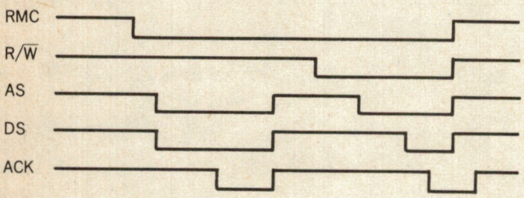
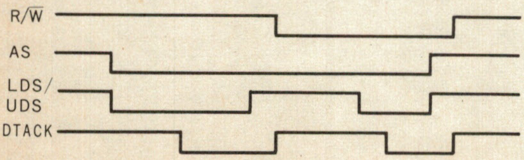


図9-9-4 他のファミリの場合のRMCサイクル



した場合のみ実現できると思われる。

以上の点から，MPUのクロックが速くなるにしたがい，等比級数的にメモリに対する要求が厳しくなる。

〔ライト・サイクル〕

ライト・サイクルについても同様の事となるが，一つ問題がある。それはリード・モディファイ・ライト・サイクルへの対応である。

ライト・サイクルの時も，アクセス・スタートをDSにたよってしまえば問題はないが，68000ファミリでは，ライト・サイクル時にはASとDSは出るタイミングが異なる。DSは，①アドレス類が出力されて，②次にASが出て，③それからデータが出て，④次にDSが出るというシーケンスを持つ。したがってメモリのサイクル自体はASから始まっていると考えられるため，DSから実際のメモリに対して書込みを行なったのではデコーダ等の関係もあって，大幅に遅くなってしまう。

そこで，68020／30ではASは各メモリ・サイクルに独立して出てくるため，メモリのアクセス自体はASを根拠に行なえる。ただしメモリ側のライト受け入れのタイミングによって，ウェイト・サイクルの採用は別に決めた方がよい。要領は，リードの時と同じである(**図9-9-3**)。

他の68000ファミリでは，先のリード・モディファイ・ライト・サイクル時（RMCと略す）にはASは連続している。そこで，ライトの疑似ASとしてライト信号とASのANDをとったものをライト時ASとして採用するか，完全にDSベースでアクセスを行なう事になる。ただし，メモリ側がRMCを受け入れられる場合には，メモリにアクセスを入れたままでDSをストローブとして使い，R／Wの信号線でバッファや他の機構を切り替えて実行できる場合もある。

いずれの場合も，アクノレッジは，DSから返す方法が非同期方式の正しい処理と言えよう。

68020／30では，以上のようにASはメモリのアクセスを表示する信号で，DSは実際のデータ転送用のストローブと言える。

他のファミリでは，ASはアドレス類の安定及び

バス使用中を示す信号で，ＤＳはストローブと実際のアクセスを示す信号と言える。特に，DTACKの作成においてはDS (UDS, LDS)をベースにしないと，RMC時にリード・サイクルのDTACKが返らずバス・エラーとなるか，Ｌレベルに保たれたままになっていてタイミングがとれない等の事態が発生する。

この他に，68020では従来の68000ファミリになかったメモリ・アクセスに関わる出力信号ＥＣＳとＯＣＳがある。

ＥＣＳは，ＭＰＵがメモリ・サイクルを始める事を示す信号で，サイクルの始めの½クロックの間だけ出力される。この信号はデュアル・ポートＲＡＭのアービトレータや，ダイナミックＲＡＭのリフレッシュとのアービトレータにアクセスがくる事を伝え，アービトレイションの準備をさせる。

ＯＣＳは，ＥＣＳと同一のタイミングで出力されるが，あきらかな命令フェッチ・サイクルでは出力されない。つまり，オペランド・サイクルの始まりのみを表わす。

9-10 メモリのアドレス・デコーダ

メモリについて特に重要なのは，アドレス・デコーダの設計である。68000ファミリでは，数多くのアドレス線，ＦＣコード，更に68020／30では，ＳＩＺコード等もデコードしないと，アクセスされるメモリの１～４バイトを特定できない。もし，68000／10で全部をデコードすると26本，68020／30なら37本のデコードが必要となる。

当然，デコーダは大きくなり，その遅延時間も長くなる。そこで，このデコードにかけられる時間の上限が存在する。この制限は，メモリ・アクセスのスタート側においてではなく，アクセスの終了と次のアクセスのスタートとの間によって決定される。

当然の事であるが，メモリ側からのアクノレッジ信号（ＡＣＫ）は，デコーダの出力から作られる。そこで，一つのメモリ・アクセスに際し，デコーダの遅れは単にメモリ・アクセスに必要な時間が長くなるだけであるが，一つのメモリ・サイクルが終わってから次のメモリ・サイクルが始まるまでの時間（クロック数）は変更できない。ところが，先のアクノレッジ信号は，次のサイクルのスタート以前(正しくはＡＳのアサート時）までにネゲイトされていなければならない。この点からデコーダの持つべき速度が決められるのである。特に68000／10／08では，ＶＰＡというDTACKに代る働きを持つ同期型のメモリ・サイクルを指示する信号入力もあるし，BERRもあるが，これら全てにおいて次のサイクルのＤＳスタートまでにアドレス・デコーダからの信号が完全に消えていなければならない。アドレス・デコーダの論理は全てＡＮＤ動作であるが，入力によって出力が安定に出るまでの時間は出力を安定に消す時間におおむね等しい。そこで，この時間をだいたい１クロック以内に納めておかないとトラブルが発生する。読者の中には，DTACKを返す時ＤＳをストローブで入れておけば問題ないと思う方がいると思われるが(筆者がかつてそうであった)，次のメモリ・サイクルのＤＳまたはＡＳがイネーブルになると，一度消えた古いDTACKが遅れ時間の分だけ再び表われる事がある点に注意されたい(図9-10-1のゴーストのDTACK)。これを防ぐには，デコード出力を適当なタイミングでラッチするか，デコードを特に高速化して逃げるかであろう。

図9-10-1 メモリのアクセス異常を引き起すアドレス・デコーダの遅れⒶと異常の発生タイミング

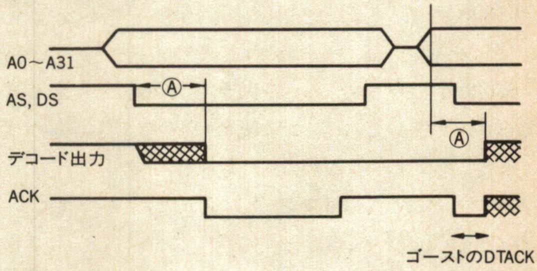
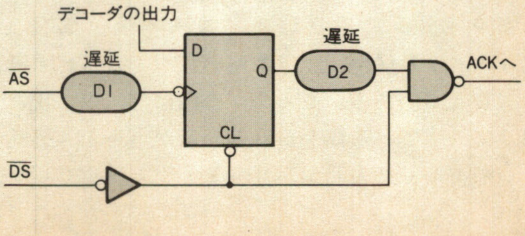


図9-10-2 デコーダの遅れを回避するための制御回路



但し，まったく効果がない訳ではない。次のメモリ・サイクルのスタートしてからＡＳの出てくるまでの１クロック弱の時間稼ぎにはなる。

図9-10-2にラッチを使った例を示す。こうするとラッチの動作はＡＳとＤＳのタイミングのみで動作するので，ゴーストから逃げる事ができる。

9-11 内蔵キャッシュ・メモリ

68020は様々な16／32ビットのＭＰＵの中でキャッシュ・メモリを持っているのが大きな特徴の一つである。

キャッシュとはＭＰＵが実行する命令や使うデータ・エリアの特に良く使われる部分を取り出してＭＰＵに直結した高速のメモリ等にこれらの部分を置いて，外部の主記憶（一般に大容量になる程遅くなる）に対するアクセスのための遅延（ＭＰＵに対するWAIT）を起こさないようにするためのものである。

キャッシュには様々な方法があるが，68020のキャッシュは一つのシリコン・チップ上に集約して乗せているオン・チップ・キャッシュという制限もあって，256バイトに制限されている。

また扱うのは実行される命令のみで，インストラクション・キャッシュといわれるものである。

68030では，さらに256バイトのデータを扱うキャッシュを有する。

68020でのキャッシュの大きさは256バイトと言われるが，これがどのように使われるのかを解説する。

まず，256バイトというのは本来の命令が入る部分のみの大きさで，64ロング・ワード（４バイト）に分けられて使われる。

68000系の命令は２バイト・ワード単位であり，もちろんロング・ワードもワード・アクセスは可能である。

キャッシュには更にアドレスの上位部分（Ａ31～Ａ８）を保持，記憶するタグ（ＴＡＧ）と呼ばれるＲＡＭがある。このタグの内には，更に現在メモリに入っている命令が有効か否か（正しく書込まれたか否か）を表わすバリッド・フラグ（Ｖ）と，命令がスーパーバイザの命令かユーザー側の命令かを示すＦＣ２を入れておく部分がある。

つまり，このインストラクション・キャッシュは**図9-11-2**のように1＋1＋24＋32(ビット)，計57ビット・ワード×64（アドレス・Ａ７～Ａ２）で構成されている（命令なのでバイト・オペランドはない。つまりＡ０はここに入ってこない)。

キャッシュから命令は，次のような手順で読出される。

1)．ＭＰＵが命令フェッチ・サイクルを起こす。

2)．Ａ７～Ａ２がデコードされてキャッシュの内容が読出される。

3)．Ｖ＝１でかつＦＣ２，A31～Ａ８の内容がアクセスの条件と一致すると，キャッシュ・ヒットと言われてキャッシュの命令記憶エリアの内容がＡ１，SIZ信号により，全体として，または上位，下位ワードのみ，インストラクション・パイプに送られる。

この時，68020は完全に命令の読出しに際してWAITのない状況で，命令を高速で実行できる。このキャッシュ・ヒットの判定は，ＭＰＵの外部への読出しに先だって行なわれるもので，この作業は非常に高速で行なわれる。もし，キャッシュ・ヒットがない時，命令読出し機構は外部への命令読出しのためのメモリ・サイクルをスタートさせる。

3)において，キャッシュの内容がヒットしなかった場合，その命令がチップ外部から読まれてインストラクション・パイプに入る。このとき，ＭＰＵはヒットしなかったキャッシュの内容を命令，タグとも更新し，もちろん（Ｖ）フラグをセットする。

したがって，キャッシュの内容はヒットがなければ次々に更新されてゆく。キャッシュがヒットされるのは，上位アドレスによらず，下位７～８ビットが異なっているループをＭＰＵが実行していく時に最も高く実現する（キャッシュのヒット率が高いと言う)。

これは，ＭＰＵがデータ・レジスタ内のデータをループで加工してゆく時などに実現される。キャッシュがヒットしていても，オペランドが外部のメモリにあって外部アクセスが起きれば，当然外部にメモリ・アクセスのための遅延が入る。

以上の動作は68030のデータ・キャッシュでも同様に行なわれる。但し，データ・キャッシュでは，アドレスとＦＣ，がヒットしての書込みサイクルに於て，キャッシュと外部メモリの両方に書込みがなされる点が異なる。

特に68020／30は外部のメモリに対する要求がシビアで，普通のDRAMなどを使えば，たちまち３～４クロックの待ちサイクルが入り，そのためにキャッシュの有効性は下がってしまう。そこでキャッシュを生かすには，短ループでの処理に外部メモリを使う比率を下げ，データ／アドレス・レジスタのみで演算する。またはＭＰＵの近くにローカルの高速ワーク・メモリを置くのが有効と思われる。

9-11-1　キャッシュのコントロール

68020／30のキャッシュはインサーキット・エミュレータや，評価のために外部からキャッシュの使用を禁止する入力ピンを持っており，キャッシュを一切禁止できる。

また内蔵のレジスタにMOVEC命令を使いアクセスする事により，全ての(Ｖ)フラグのクリヤ，キャッシュの内容の現在値保持(書込みのみ禁止)，使用禁止ができるようになっている。そのコントロール・ビットは**図9-11-3**のようになっている。

〔注意〕このキャッシュ・メモリの内容へのソフトウェアからのアクセスはできないし書換えも，もちろんできない。したがって内容を知る事はできない。

またキャッシュがヒットしている間は，ＭＰＵは外部への命令フェッチ・サイクルを起こさない。したがってＭＰＵのバスは空白期間（アイドル状況）が多く発生し，ディスクと主記憶との間のＤＭＡや他のＭＰＵ（同一バス上の）アクセスのチャンスが増え，トータルでのシステム性能を更に良くする。

但し，以上のような性質のため，プログラムのインサーキット・エミュレータによるリアル・タイム実行は，まず不可能と言ってもよい（一般にインサーキットではキャッシュを禁止している)。また命令のサイクル数を個別に数え実行時間を推測する方法も，パイプ・ライン処理と伴って不可能に近く，実行してみるまで実行時間の正しい値は判らなくなっている。

図9-11-1　キャッシュの働き

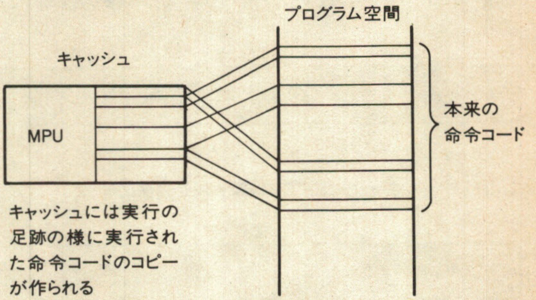
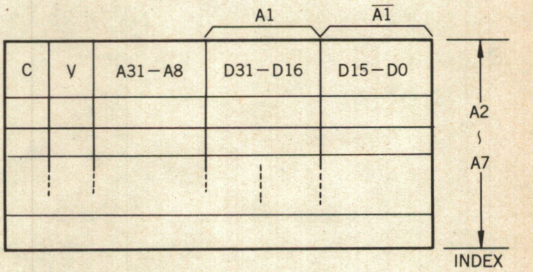


図9-11-2　キャッシュ・メモリの構造



9-11-2　CAAR (Cache Address Register)

これもMOVEC命令によって読出されるレジスタで，ＣＥビットによって処理されたアドレスがこのレジスタにラッチされている。

9-11-3　CCR (Cache Control Register)

キャッシュ・メモリをコントロールするためのレジスタは一応32ビット長あり，MOVEC命令によってアクセスされる。このレジスタは68020では下位４ビットのみが意味を持ち，68030では，下位14ビットが使われている。他は全て０となっている。

〈ビット０〉ENABLE

このビットはＭＰＵにキャッシュの利用の許可，不許可を決めるビットである。

リセット時には０となり不許可となり，キャッシュを利用して良ければ１を立てる。０のままではＭＰＵは全て外部のメモリをアクセスし，キャッシュの内容も更新されない。

〈ビット１〉Freeze Cache

キャッシュの内容の書換えを禁止する（キャッシ

図9-11-3　キャッシュ・コントロール・レジスタの構成

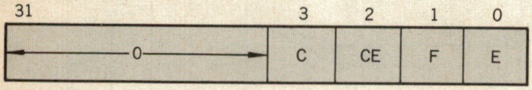
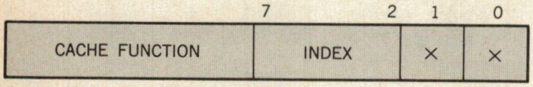


図9-11-4　キャッシュ・アドレス・レジスタの構成



ュに内容がなければ記入される)。ただしキャッシュの内容にMPUの読出しがヒットすれば外部アクセスは起こらず，キャッシュの内容が命令デコーダのパイプに送られる。

〈ビット 2〉 Clear Entry

このビットは読めば常に0が読まれるビットだがここに1を書くと書いた時のCAARのINDEXの値に対応する（V）フラグを0にしてその内容を無効とする。この動作は実際にキャッシュがヒットしたか否かには無関係で，CAAR内の上位のアドレスがキャッシュのタグと一致していてもいなくても実行される。更に，この動作はこのレジスタ内の他のビット(E)，(F)や外部のキャッシュ禁止入力のCDISの状態とも無関係に行なわれる。

これにより，キャッシュ内の特定の内容を無効にし，どこの内容を無効にしたかはCAARから読出してチェックできる。

〈ビット 3〉 Clear Cache

このビットも読出すと0にしか読めないビットだが，1を立てる命令を与えると，キャッシュ内の全ての（V）フラグがクリヤされ，よって全てのキャッシュの内容が無効とされる。これによってＯＳとユーザー・プログラムの切替え時等，ＭＰＵのソフトウェア環境が変化する時の対応を良くする事ができる。

68030では，このビット0～3の4ビットに対応するデータ・キャッシュ用のコントロール・ビットが8～11ビットに存在する。これに，命令，データの各々にバースト・イネーブルのビット（4，12）とライト・アロケイト（4）のビットが追加された。バーストはキャッシュ内のデータ／命令をブロック転送で入れ替えるのに用いられ，ライト・アロケイトはデータ・キャッシュ内のアドレス・タグの入れ替えの禁止／許可を決定する。

この中で，特に多用されるのは，ＣＣ(クリヤ・キャッシュ)である。これは，ユーザー・プログラムからＯＳへ，またはＯＳからユーザーへとプログラムの実行環境が変化する時，特にMMU（メモリ管理ユニット）を持っているシステムでは重要である。なぜなら，MMUにより，プログラムの実行アドレスがユーザー1側とユーザー2側で同じになっている可能性があるからである。このような場合，メモリ・アクセスは必ず，まずキャッシュを，次にキャッシュに該当するものが入っていない場合に外部メモリをアクセスにいく。実行環境が切替ってキャッシュの内容が意味を持たないものをそのままにしておくとＭＰＵは，時々，ある外部のプログラムを読むべき所を，まったく無関係の内部キャッシュの古いプログラムを読み実行してしまう。これは，たまたまその実行アドレスをキャッシュのタグの中に持っていたため，外部の新しいプログラムはキャッシュ側に優先権があるために無視され，キャッシュ内の現在では意味を持たない命令を実行する事になる（図9-11-5)。このようなコンテクスト・スイッチに関わる実行ミスを避けるため（コンテクスト・スイッチとは，プログラム全体の実行環境がＭＰＵから見て切替えられてしまうような実行環境の変化を言う)，ＣＣをコンテクスト・スイッチを行なう直前に実行し，内部のタグを全て無効にしてやる必要がある。またコンテクスト・スイッチによって切替えを受入れる側がもし可能ならＣＣを行ない，前のコンテクト（プログラムの）内容を全てクリヤしてから実行するのが最も望ましい訳である。

この処理で最も多いのは「ＲＴＥ」命令によるＯＳ側からユーザー側へのリターンだが，この「ＲＴＥ」の直前に実行してやると良い(図9-11-6)。

キャッシュは，プログラムの実行速度の点では大変効率が良くなるが，MMUを用いたシステムでコンテクスト・スイッチを行なった時には，このような欠点もある事を予め知り，対応したシステムを作ることが望ましい。

図9-11-5　MPUはキャッシュをまずアクセスして有効でさえあれば，それを実行する

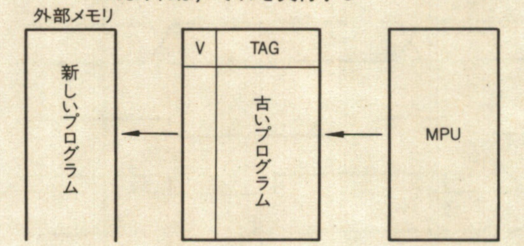
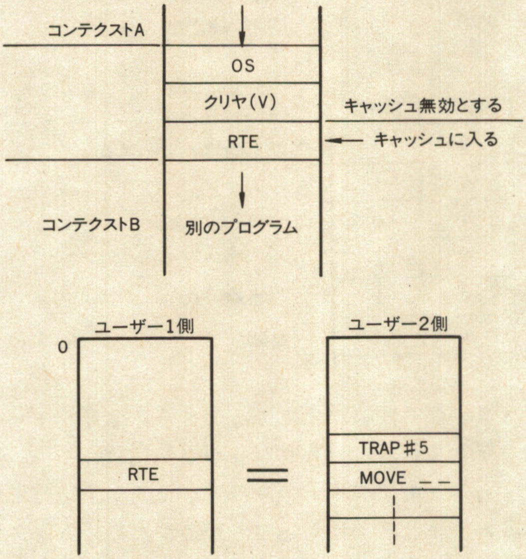


図9-11-6　コンテクスト切替時のキャッシュ・コントロール



9-12　ダイナミック・バス・サイジングについて

このダイナミック・バス・サイジング(DYNAMIC BUS SIZING, 以下ＤＢＳと略す)は68020のハード上の最大の特徴の一つであり，最新の32ビットＭＰＵとしての特徴を示すものである。

ＤＢＳとは一口に言えば，ＭＰＵに接続されるデータ・バスの幅がバイト単位で１，２，４バイトの任意の幅で使う事ができ，かつメモリのブロックごとにこの幅を適応させる事ができるアーキテクチャと言える。

その方法は，68000ファミリのＭＰＵはメモリのリード・ライトの際に，データ・ストローブに対してアクノレッジを返す方式を基本としているが，このアクノレッジを返す時に，現在アドレスされているメモリの幅が何バイトであるかを含むアクノレッジを，メモリ側からＭＰＵに返してやる事により実行する。この時，この信号はメモリ側のレスポンスに充分なタイミングでＭＰＵに与える必要がある。

また特に注意すべき点として，何バイトのデータをメモリとの間でやり取りしようとしているのかに関係なく，現にアクセスされている所のメモリの幅で返してやる事である。

何バイトのデータを転送しようとするかという事に関しては，68000のファミリでは，Ａ０とデータ・ストローブＤＳの合成信号であるＬＤＳ，ＵＤＳが使われており，Ａ０はない。ところが68020／30では，アドレスはアドレスとしてＡ０～Ａ31まで32本全てが存在し，バイト単位のメモリを指示し，データ・ストローブはＤＳとして独立して存在する。そして新たな信号としてSIZ0, SIZ1の２本のサイズ信号がある。この信号により，各メモリ・サイクルの転送バイト数を指示している。

実際にアクセスされるメモリは，これらA0,A1, SIZ0, SIZ1によって決定されるが，アクノレッジ信号はこれらの信号とは無関係に，アクセスされているメモリに与えられている幅によって決まり，適当なタイミングで68020／30に与えられ，１メモリ・アクセスは終了する。この様子を**図9-12-1**に示す。

図9-12-2は，68020／30のアドレス空間中に配置されたメモリとそのアクノレッジの関係を示している。

（注）こうした例でも理解できるかと思うが，サイズ信号は転送要求のバイト数をそれが可能か否かに関わりなく示し，アクノレッジはそれが使われているか否かに関係なく，アクセスされている所のメモリの幅信号を返している。

そして，返されてきたメモリの幅の内容によってＳＩＺ信号は新たに作られ，転送バイト数が０になるまで次々とメモリをアクセスしていく。この時，１データ転送において，ＳＩＺ信号の指示する幅は徐々に狭くなってゆく事になる。

この項の整理として，ＤＢＳの信号とＳＩＺ信号との関係を以下に示す。

図9-12-1　メモリの幅とDSACKの関係タイミング図

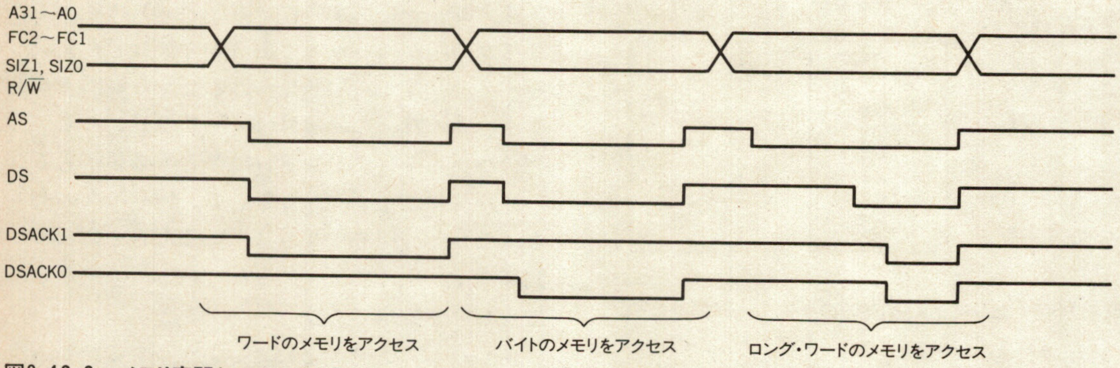
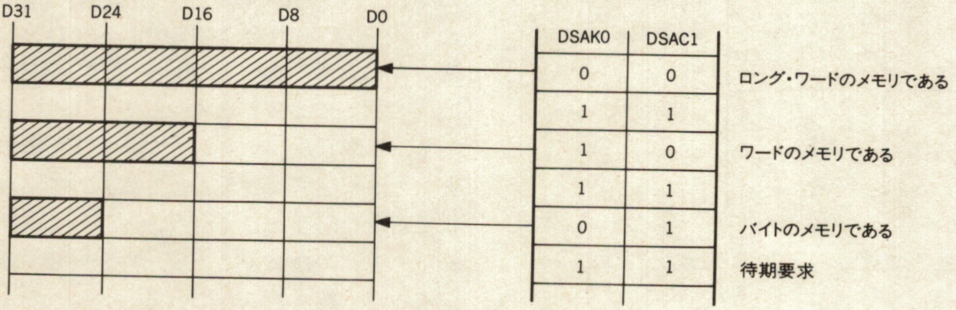


図9-12-2　メモリ空間とDSACKコード



1)．MPUが転送を要求するメモリのアドレス(A０～A31）と転送したいバイト数をＳＩＺ信号として出力し，同時に転送の方向をリード/ライト(R/W)で示し，ＦＣコードも出力する。

この時，更にＣＰＵがリード･モディファイ･ライト命令を行なうのであれば$\overline{\text{RMC}}$信号が出力される。

2)．次にＡＳ信号がロー・アクティブとなり，転送のための諸信号が安定した事を示している。

3)．アドレス，ファンクション・コードによって指定されたメモリ・ブロック（Ｉ／Ｏの場合も同じ）は以下の作業を行なう。

Ａ1，Ａ0の内容により，どこのメモリからアクセスされるのかをデコードしＳＩＺ信号により何バイトがアクセスされるのかを設定する。

例を示すと，1バイト幅のメモリであればＡ1，Ａ0は全てデコードに使われる。そしてアクセスがあるという事は，サイズは1以上のためデコードの必要はない。もちろんDSACK信号はバイトをＭＰＵに指定する。

メモリの幅が2バイトの時は，ＳＩＺのデコードとＡ0の間に関係が出てくる。ＳＩＺ信号が1バイトを示している時はＡ0が0か1か(偶数か奇数か)によって，上位バイトのみのアクセスか下位バイトのみのアクセスかに対応する必要が出てくるからである。この場合はアクセスのコントロールが**図9-12-3**のように8種類に分かれる。A0=1の時の動作は全て同じで，A0=0でバイトの時のみ上位バイトがアクセスされるという特徴がある。この時$\overline{\text{DSACK}}$の信号はどの場合でも，ワードでMPUに返す。**図9-12-3**で，転送未了のバイトが残る場合はMPUのメモリ・コントロール機構は次のメモリ・アクセス時に，ＳＩＺ信号を計算しなおし所定の転送を行なってくれる。

ロング・ワードの場合は，かなり複雑な制御がメモリ側に必要とされるが，その様子を**図9-12-5**に示す。

このように全ての場合においてＡ0，Ａ1によってメモリ・ワード内のオフセットを決め，ＳＩＺ信号によってそこから何バイトを実際にアクセスするかを決めると考えられており，MPUのソフトウェ

アから見たメモリ空間は，完全にバイト・イメージとなる。

9-12-1　ビット・フィールド命令との関係

ビット・フィールド命令は，1～32の任意長のデータを扱い，もちろんメモリとの間のやりとりもある。この場合，DBSがサポートできるのはバイト単位であり，ビットではないので，MPUはこれらの命令をリード／モディファイ／ライトの動作を通じて行なう。つまり，MPUは必要なだけのデータを読込んで，そこから必要なビットの範囲を取出し(リードのみなら取出して終わり)，変更してメモリに戻す。この時，データ長が1～32ビットであっても，1ロング・ワード内に納まる場合には，1リード／モディファイ／ライトで終了するが，オフセット計算の結果によって2回このリード／モディファイ／ライトが必要とされる。この場合は，実行に要するサイクル・タイムも上るし，メモリ・バスの使用回数も倍になる。これは最大3回にまで分割されるのは通常のロング・ワード・アクセスに等しい。

〔注〕ここでの動作は，リード・モディファイ・ライトの動作であっても，リード・モディファイ・ライトのメモリ・サイクルではない。したがってRMC信号は出力されない。またリードとライトの間に別のメモリ・サイクルが入る事もある。

9-13　68000ファミリと同期型I／O

68020／30にはないが，他の68000ファミリには入出力ディバイスとして8ビット系の同期I／Oチップを使うための信号「E, VMA, VPA」を持っている。「E」はMPUクロックを1/10に分周し，4クロック間Hレベル，6クロック間Lレベルを出力し，命令の実行，リセット，ホルトにも無関係である。

「VPA」は非同期指定のタイミング信号でDTACKの代わりとしてMPUに与えられると，MPUはEとの同期関係をチェックして適当な所で「VMA」を出力してくる。VMAが出力された所で，同期型の入出力素子をアクティブにしてEの立ち下りでデータを転送する。

図9-12-3　SIZコードとA0, A1によるメモリのアクセス幅及び個所の関係(ワード・メモリ)

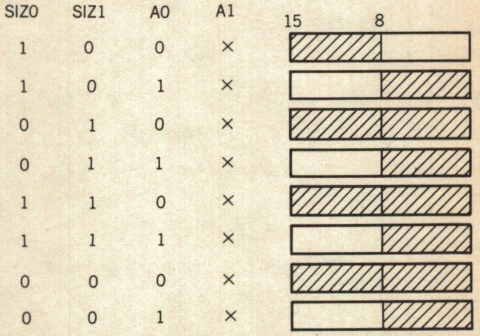


図9-12-4　A0＝1, サイズ＝バイト時には，メモリ側でバイト・データのみアクセスされるようにする

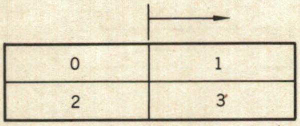
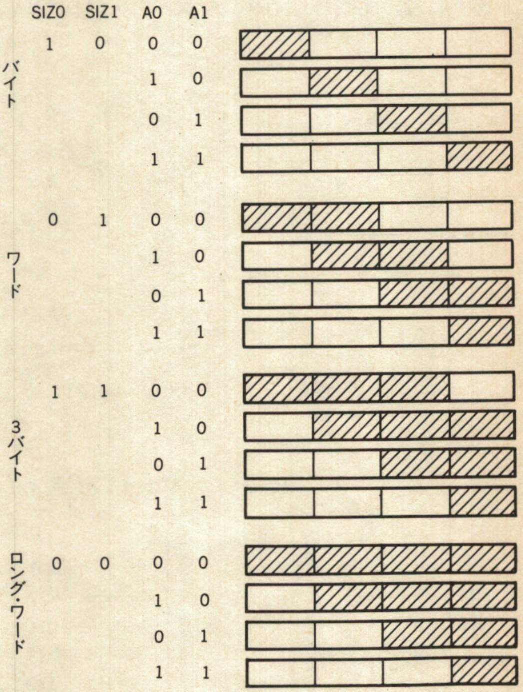


図9-12-5　SIZコードとA0, A1によるメモリのアクセス幅及び個所の関係(ロング・ワード・メモリ)

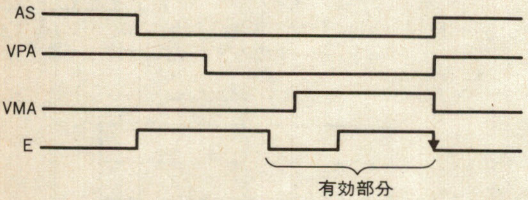


68000ファミリの入出力素子の大半は，このタイミングですぐ接続できるが，80系の素子はこのままでは接がらない。図9-13-2のような回路でRDストロ

ーブ，WRストローブに変更すると，そのまま接続できる。実用的には，**図9-13-3**のようにデコーダを用いたものが使われる。

両者ともCS信号は必要な上位アドレスをデコードしたもので，この信号は，VPAとして返されている。ただし，通常の68000系のI／Oとはデータの保持時間等が若干異なるため，微妙な使い方の場合には注意を要する。

図9-13-1　同期形メモリ・サイクルを使った場合のタイミング



データ転送は矢印の時点で行なわれる

図9-13-2　80系のI／Oディバイスへのタイミング発生器の考え方

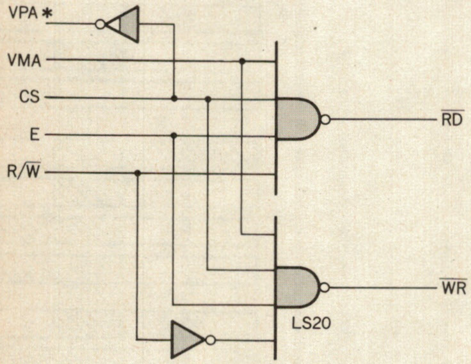
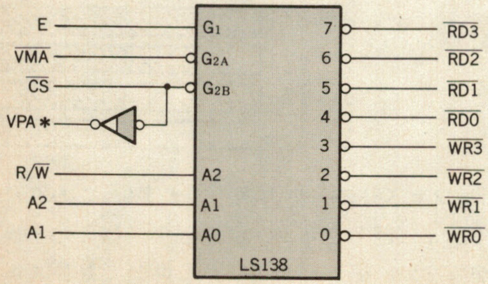


図9-13-3　80系のI／Oディバイスへのタイミング発生器（実用例）



9-14　メモリ・システム設計の盲点

私の失敗談になるが，完璧な設計をしたつもりのメモリ・システムにも落とし穴があるという一例である。何事も先入観にとらわれると思わぬ失敗をするケースと言える。

設計した68000のシステムは，8MHzクロックでは何ら異常がない。しかるに12MHzでは動作しないということがあった。

このシステムのメモリは，アクセス・タイムの制限上，12MHzの時は，正しいウェイト・サイクルを入れてあった。が，メモリのアクセス時に不特定のエラーが発生しシステムを停止させた。

クロックを8MHzにしてメモリ・サイクルを観測しても異常は見られず，余裕も充分とっていたのだが12MHzでは動作しない。

最終的にはロジック・アナライザで原因を見つける事ができたのだが，VPAのネゲイトが遅れ，次のメモリ・サイクルのスタートにかかりかけているために起きた異常動作だったわけである。

この時，観測されたのは，入出力の初期設定のため，VPAをMPUに与える同期式のメモリ・サイクルがあり，次にダイナミックRAMのサイクルが続くのだが，この同期式のサイクルは当然正しく終了し次の非同期サイクルも正しくDTACKで終了しているように見えるが，このサイクルの始めに一部VPAの信号が残っていたのである。

すると，同期式のサイクルからは二つ目のサイクルがDTACKを与えても反応せず，同期式信号のVMAを出力してきて異常となり，そのまま若干暴走し，ダブル・バス・フォールトになって止まるというものであった。

ここでの失敗は，メモリのアクセスにのみ注意が集中して，ネゲイトは当然の事として注意不充分だった事である。68000ファミリは，アドレス・デコードのライン数が多いために，うっかりするとこのアクノレッジ系の信号のネゲイト条件を守れなくなる。特に，DSACKxxやDTACK，VPA，AVEC等の信号にワイヤードORを採用して，システムの拡張性を

大きく取ると，拡張した部分で分布容量等が増しネゲイトがC充電のダラダラとして波形を描きスペックを守れなくなる事が多いからである。

例として，68000ファミリのDSACKのネゲイト条件を**表9-14-1**に示す。これはAVEC, BERRも同じタイミングが要求されており，次のメモリ・サイクルのＳ０の範囲内に納まっていなければ，いけないという制限をとる事ができる。

またシステムを設計する際，ワイヤードＯＲの数を多くとらず，できればゲートでＯＲをとった方が良い結果が得られる。

9-14-1　ホルト信号の注意点

これは68000ファミリ全般にわたる注意事項であるが，リセットとホルトの信号は双方向性となっている。

特に，HALTが内部からドライブされる場合は，ダブル・バス・フォールトを検出して，ＭＰＵが停止した時であるから，この信号が出ている時は外部からの介入が必要である。そこで**図9-14-2**のような回路によれば内部発生のホルトの検出ができ，外部にＣＰＵ停止を知らせる。

9-15　ＭＰＵのリセット・スタート

68000ファミリのＭＰＵのリセットからプログラム・スタートの手順を以下に示す。リセットは何時でも受けつけられ，命令またはメモリ・サイクルの区切れにも無関係に実行され，バス・エラー等の他の例外処理と異なりスタックへの内容セーブも行なわないため，回復は不可能である。

またダブル・バス・フォールトによって，ＭＰＵが内部発生のHALT状態にある時の回復は，リセットしかないため，何らかの方法により外部から行なう必要がある。

9-15-1　リセットを与えるハードウェアの必要点

1)．電源投入時

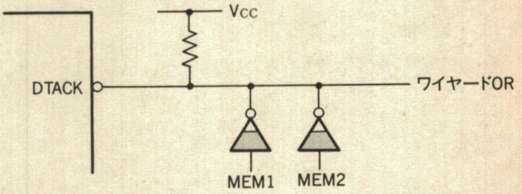
電源投入時には，ＭＰＵの内部のダイナミック動作をしているレジスタ類のプリ・チャージのために電源電圧が安定してから100msec以上，リセットをＬレベルに保つように要求されている。

2)．電源投入時以外

10クロック・サイクル以上与えられれば，即，動作する（ただしReset命令実行中以外の場合)。この場合も含めるならば，68020では最低520クロック・サイクル，他のファミリでは140クロック・サイクルほど必要になる。

このリセット信号は，68020ではリセット信号のみで完全に動作するが，他のファミリではシステム全

図9-14-1　ワイヤードORによるDTACKの返し方



MEM1, MEM2は各システムごとに別々のアクノレッジを出す

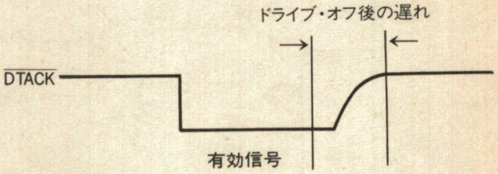


図9-14-2 HALTドライブ回路例

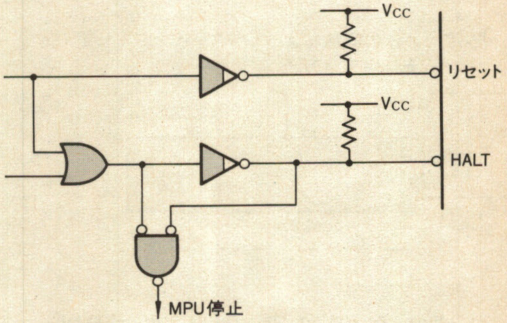


表9-14-1　68020の場合のACKのネゲイト許容時間

MPU TYPE	DSネゲイトよりDTACKネゲイト(MAX)
68000-8MHz	245nsec
68000-12MHz	150nsec
68020-12MHz	110nsec
68020-16MHz	80nsec
68020-25MHz	40nsec

体をリセットする場合にはHALT信号を同時に与えるようになっている。ただし単に再スタートさせるだけであればリセットのみでも動作する。

このリセット信号はすみやかな立上げ，つまり信号の解除が要求されており，68020では2.5クロック以内，他のファミリでは0.2μsec以内となっている。

リセット期間中にＳＲ内のＳフラグはセットされトレース・フラグ，マスタ・フラグはクリアされる。更に，割込みのマスク・フラグはレベルが７にセットされ，ＮＭＩ（レベル７割込み）以外の割込みは禁止される。

ＶＢＲは０にセットされ，例外処理ベクタ・テーブルは０番地から始まる。

リセット信号がネゲイトされると，ＭＰＵはスタート時のプログラム・カウンタの値と，スーパーバイザ・スタックの値（68020／30ではＩＳＰ）を，ベクタ番号１と０から読出す。

図9-15-1　ベクタ番号0，1をRAMからプログラムROMにアクセスのすり替えを行なう方法

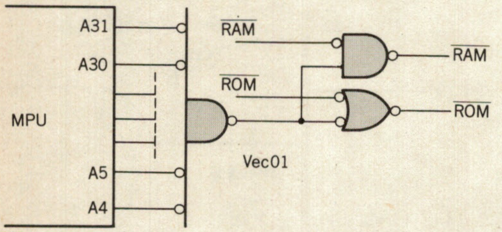
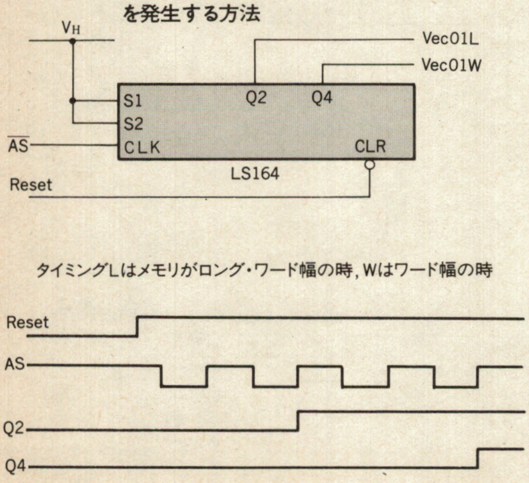


図9-15-2　ASの回数によってベクタ0，1のすり替え信号を発生する方法



この０番地附近は一般にはＲＡＭエリアとなっていて，プログラムの進行に伴ってベクタを書換えて各種の例外処理に対応する。このためスタート・ベクタとスタック値を拾う０～７番地も，ＲＡＭになっているのが一般的である。

そこでシステムのモニタまたはブート・ローダ，立上げ用のイニシャライザ等が入っているＲＯＭが必ずシステムには実装されているはずであるから，そのＲＯＭエリアの一部のイメージとして０～７番地のアクセス時にアクセスされるようにして，このスタート・ベクタとスタック値を与える。

そのため，０～７番地をデコードしてＲＡＭのアクセスを禁止し，ＲＯＭの方にアクセスが入るように工夫するのが一つの方法であり（**図9-15-1**），もう一つの方法は，リセットでクリヤされるカウンタによってＡＳの数を数えさせ，ＲＯＭのバイト幅によって２バイトなら４回，４バイトなら２回のアクセス期間中のみＲＡＭを禁止して，ＲＯＭを生かすようにする（**図9-15-2**参照）。

ベクタ・テーブルのアクセス時，この０と１のベクタ番号を持つスタート時スタックとスタート時ＰＣの値は，プログラム（それもリセット直後のためスーパーバイザ）のＲＯＭに記入されている。そこでＭＰＵは，ベクタ・テーブルのアクセス時にスーパーバイザ・プログラムのファンクション・コードを出力して，プログラムＲＯＭのアクセスを正当化している。

これ以外のベクタ・テーブルのアクセスは，全てスーパーバイザのデータのファンクション・コードが出力されるし，プログラムのオペランドとして書直す必要があるため，スーパーバイザ・プログラムのオペランドとなるスーパーバイザ・データのファンクション・コードでよい。

もし，このベクタを読出している最中にバス・エラーが起こると，プロセッサはダブル・バス・フォールトとして内部からHALTをドライブして，全ての処理を中断してHALT状態となり新たなリセットがくるまで待期となる。

0と1のベクタ番号のベクタが正しくアクセスでき，ＰＣとＳＳＰにセットされると，プロセッサは次にＰＣの番地をアクセスし，そこから読まれた命令は命令パイプに詰込まれる。パイプに入った命令は順に送られて実行を始める。こうしてプログラムによるシステムの初期化を行なう。

この時，MMUを持っているシステムでは同時にMMUもリセットされ，パス・スルーのモードになっていなければならない。そうしないと，このベクタ番号0や1の読取りに影響が出たり，絶対番地で指示されているこのベクタの指定先が変わってしまう事になってしまう。

9-15-2　クロック

68000ファミリでは，全てＴＴＬ入力レベルのクロックを必要とし，その周波数は8MHz～33MHzにわたっている。ＭＰＵのタイプと動作周波数を選ぶことにより，この外部から与えられるクロックの制限が決まる。68000ファミリのＭＰＵは全て内部にダイナミックな要素を持っているので，動作する最低周波数があり，68000では1/10，68020/30では1/2となっている。

また，特に注意してほしいのは，クロックのデューティ比で，クロックのＨレベルとＬレベルの比が20～30％もずれると(これ位は通常のTTLゲートを通すと容易に変わってしまう)，1クラス上のクロックが使えるＭＰＵを選ぶ必要がある。ＭＰＵのクロックは，フリップ・フロップの出力等，デューティ比が50％になっている所からとる事をすすめる。

9-16　ＭＰＵの実行速度

ＭＰＵの実行速度は，68000ファミリではＭＰＵのクロック数にほぼ比例して上昇する。これは当然の事である。

問題はメモリのウェイト数がどの位影響を与えるかだが，私の所にある実験用のワンボードＣＰＵで実験したところ，特別のプログラムでない単に数を数えるようなプログラムでは，ノーウェイトに対して1ウェイト入れた場合はほぼ25％実行時間が多く必要となった。これは68000の通常のメモリ・サイクルが，リード／ライトとも4クロック・サイクルなのが1ウェイト・サイクルを入れたため，両方とも5クロックになった場合，このメモリの遅延時間に完全に比例していた。この実験の結果からすると，単純なアドレス・モードを使い，1命令の時間が特に長くなっていない限り，命令読込み機構は休みなく働き，命令パイプと命令のオペランド・アクセスの同期のため，命令の実行時間に待ち時間が直接影響したものと思われる。

もちろん，この実験はプログラム，オペランドとも，同じメモリ上に乗って連続的にメモリ・オペランドを書換えるというものであるため，これほどはっきりと影響が出たのであろう。

実際のプログラムではＰＣ相対のアドレス・モードや，68020／30のメモリ間接や，インデックス付間接のモードなどでは〈ea〉の計算にかなりのサイクル数を必要とする。一例をあげると，68000でインデックスとオフセット付の〈ea〉を計算するのには10サイクルかかる。この間に2回メモリ・リードが入るがこのメモリ・リードはおそらく1ウェイトを入れても10サイクル中に吸収されて，余り外に出てこないのではないかと思われる。

特に乗除算命令のように，数10サイクル以上かかる命令の場合，数ウェイトが入っていても，まったく影響が出ないと思われる。このようにプログラムの内容によってかなり実行時間は変わり，そのためメモリのウェイト数の影響も一様とは言えない。

別の例であるが，実際には，むしろプログラムの組み方の方がより大きく影響を与えると思われる一例を示す。私の同僚が実験したのであるが，8190までの素数を求めるプログラムを実行させ，その所用時間を計った。その結果が113ページのベンチマーク・データで，68000ファミリの場合，オペランドの形式に大きく影響されており，その値は10％～20％などでなく何倍かになってしまう。

この例で判るのであるが，もう一つのメリットは68020のキャッシュ・メモリの有効性である。特にこのプログラムはキャッシュの中に収納できる程度のループを持ったため，外部のアクセスのなくなる事

により，このように良い結果が得られたのだと考えられる。

9-17 68000ファミリの実行時間

8ビット系のMPUの実行時間は，各命令の実行サイクル数と1サイクルの時間から容易に計算できた。ところが，命令プリフェッチ機構を持ち，更に命令キャッシュまで持っている68020／30の実行時間はどうか？　はっきり言えば，精密に予想する事は非常に難しく，だいたいの予想しかできない。

68000ですら命令先読み機構と実行サイクルとの関係で，計算値からはずれる事もたびたびあったため，68020／30では当然と言える。

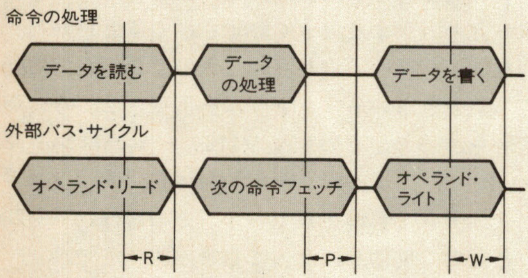
68000ファミリのユーザーズ・マニュアルを見ると，命令の実行時間は次のように示されている。

68000の場合……16（1／1）

68020の場合……16（2／3／1）

カッコの内の意味では，MPUが外部に起こすメモリ・サイクルの数で，68000の場合は(R/W)で，Rの命令のフェッチ・サイクルを含めたリード・サイクルの数，Wのメモリ・ライト・サイクルの数で68020の（R/P/W）は，Rはオペランドのメモリ・リード・サイクル，Pはプログラムのフェッチ・サイクルの数，Wはメモリ・ライト・サイクルの数である。なぜこれが必要かと言えば，68000ファミリは，ハンド・シェーク・バスを持っているため，対象とするメモリによって1メモリ・サイクルに要する時間，つまりサイクル数が変わる。そこで，この数字とシステムに使われているメモリの種類からメモリ・サイクルに入るウェイト・サイクルの数がどれ位か予想する必要がある。

図9-15-3　68000の実行時間



Rはメモリ・リードのためのウエイト時間
Wはメモリ・ライトのためのウエイト時間
Pは次の命令のプリ・フェッチのための時間で起きたウエイト時間。このPの部分は，次の命令のフェッチ時間を短くするが，評価は難かしい

特に68000の場合，Rの数，つまり，リード・サイクルの数をプログラム部分とオペランド部分に分けて計算する必要がある。例えば，プログラムは平均1.1ウェイト・サイクルが必要で，オペランドは1ウェイト・サイクル，ライトは0ウェイト・サイクルでは，……となった時に，全サイクル数は以下の式で求められる。

全サイクル数は＝16＋(R＋1.1＋P×1＋W×0)
＝16＋(2×1.1＋3×1＋1×0)
＝16＋2.2＋3
＝21.2〔先の16（2／3／1）の例〕

ここで，平均ウェイト数1.1とは，ダイナミック・メモリを使った場合などにリフレッシュ・サイクルのため，あるアクセスの場合のみアクセスに時間が余分にかかるため，これを充分長い時間で平均化した場合の数字である。

このように各命令単独のサイクル数は計算できるが，68000ファミリでは命令の読取り（フェッチ）と実行が同期していないため，ある命令を実行中に次の命令をフェッチする。ところが，このフェッチにウェイト・サイクルが入りこむと，現在実行中の命令のステップに影響を与え，この命令の実行とメモリ・サイクルと二重にウェイトを与え，実行時間を更に伸ばしてしまう事もあり得る。

また68020では，命令キャッシュを内蔵しているため，Pの部分がキャッシュから読取れるか，または外部メモリから読取られるかは，命令キャッシュの性質上ダイナミックに変化し，プログラムの状況が変わればこのキャッシュからの読出し率(ヒット率)が変わり，平均実行時間は変化する。

したがって，精密な実行時間は実測してみないとなかなか判らない。また更に68000ファミリのもう一つの難しさは，各命令でMPUがエフェクティブ・アドレスを計算する際に要する内部の計算サイクル

を，各命令のアドレス・モードによって加算しなければならないことにある。

このアドレス・モードによっては，オペランドのデータを命令ワードのポスト・ワードからフェッチするため，命令と同様に(R/W)または(R/P/W)を持っており，命令によりオペランドの二つの実行アドレス各々について，このサイクル数を計算し，命令サイクル数と全部のオペランドのアドレス計算サイクル時間を加算する必要がある。

更に68020／30では，ユーザーズ・マニュアルに記載のあるベスト・ケース／ティピカル（標準）ケース／ワースト・ケースがある。これは，命令パイプ内での予備動作により，各命令の部分が重なりあって，その命令を見た時の所用サイクルが変化して見える事を指している。これは内部のアドレス演算時間中に，次の命令のフェッチが済んでしまえば，命令フェッチの時間は，前の命令の内に吸収されてしまう(ベスト又は標準ケース)。これとは逆に，ブランチ直後の命令は，インストラクション・パイプが空になっているので，この様な多重化は全く期待できず，逆に，後の命令のある部分を負ってしまう可能性がある(ワースト・ケース)。

68000ファミリ・ベンチマーク・データ

68000ファミリのベンチマーク・テストをするために以下のハードウェアを用意した。

● CPUボード：マルチバス形式のCPUボードで，MPU，1M・DRAM，128K・EPROM，2CHシリアル・インターフェース，フロッピーディスク・インターフェースを装備している。

● OS：OS-9/68000(V1.2)

● コンパイラ：C言語

● ソフトウェア：素数を求めるプログラムとした。演算内容は加算とフラグのセット/クリア，フラグによる条件判断のみの単純なものとしてある。

● 実行環境：68000/10はピンコンパチブルなのでMPUのみを差しかえた。68020は，第14章で紹介するアダプタ・ボードを利用して同一ボードで実行させている。プログラムはメモリ中にロードしておき，スタートとエンドのメッセージ間を計測し実行時間を割り出している。この状態で，メモリ(DRAM)をオペランドとした時は1waitサイクルが入り，1メモリ・サイクルは500nsecとなっている(クロック10MHz)。

■結果に対する評価

68000は内蔵レジスタが多数あるが，このレジスタ宣言は，各変数パラメータをこの内蔵レジスタに優先的に置いてゆく様にする宣言で，これを行なうと各変数がメモリではなく，レジスタが割り当てられる。これで各命令のオペランドがメモリからレジスタに変わり結果としてメモリのアクセス頻度が下がり，実行速度は向上する。この点では68000も68010も大差はなく各命令の実行サイクルが短縮されている分だけ10が速くなっている。特に20でキャッシュ・メモリを使った場合は，プログラムの実行部分の大半がキャッシュ・メモリに格納され，オペランドはほとんど内部レジスタとなり，外部へのメモリ・アクセスはフラグ・バイトの時のみとなるので飛躍的に速度が向上している。レジスタ宣言なしの場合でも，各命令のフェッチ・サイクルがない分(約半分)，速度が向上し，68000/10のレジスタ宣言有りの場合とほぼ等しくなっている。逆にレジスタ宣言もなくキャッシュ・メモリも使わない場合には，68020といえどもメモリの幅16ビット，メモリ・サイクル500nsecの同一条件下では68000/10と実行速度に大きな差は生じないことがわかる。

参考として，アポロ・コンピュータDN660で同じプログラムを実行させた結果を付記する。ただし，このコンピュータは外部からの計測をする事がむずかしく必ずしも同一条件とはいえないが，大型ミニコン並みの演算能力を持つマシンでの実行結果として見ていただきたい。

更に，同じプログラムをアポロDN3000で実行した結果も併載する。DN3000はMPUとして68020を採用し，32ビット・バスを使っている。OSはアポロのAEGIS，コンパイラもアポロのPASCALなので，先OS-9の場合のように68000/10/20で，まったく同一の命令コードを実行した場合との差異はあるが(クロックも12.5MHz，または16MHz)，68020の高性能は，良く理解いただけるものと思う。

■テスト方法：Sieve(1～8190までの間の素数を求めるプログラム)を10回繰り返す

CPUタイプ	キャッシュ	レジスタ宣言	時間(秒)	言語	OS
MC68000		有	2.4	C	OS-9
〃		無	4.8	〃	〃
MC68010		有	2.2	〃	〃
〃		無	4.7	〃	〃
MC68020	有	有	0.95	〃	〃
〃	有	無	2.4	〃	〃
〃	無	有	2.3	〃	〃
〃	無	無	4.3	〃	〃

※10MHzクロックのシングル・ボード・システム(OSはOS-9)にて測定。68020は，第13章のアダプタで測定。

CPUタイプ	キャッシュ	レジスタ宣言	時間(秒)	言語	OS
DN660オリジナル	?	×	0.78	PASCAL	AEGiS
DN3000，MC68020	有	×	0.62	〃	〃

※DN3000のCPUクロックは12.5MHz。

68020／30のシステムへの適用

68020を使ったシステムを計画する時，または，あるシステムに68020を使用する時には，このMPUの特性に合わせた設計が必要となる。

その要求する特性の第1はメモリのアクセス・タイムである。MPUをいくら高速にしても（現行の最高クロック周波数33MHz）メモリのアクセスに時間がかかるのでは，MPUの高速性が無意味になってしまう。もし，MPUに対し，メモリのアクセス・タイムによる遅延を無しにして動作させようと思えば，最悪値としては30nsecでメモリはレスポンスできねばならず，1クロックずつを各メモリ・アクセスによってロスしたとしても（これ位なら命令パイプによって吸収され影響は余りでない），70nsec位が限界である。ところが，このMPUはこれに反する条件として，32ビットという広いアドレス幅と更に3ビットのFCコードを持っている。これを全てデコードすると，通常の方法では30～50nsecも必要となり，効率の高いシステムの設計は困難となる。

また，このアドレス・デコーダの遅れれは，DSACKnのネゲイトの遅れ，次のメモリ・サイクルの異常動作の発生という悪循環を発生させる恐れもある。そこで，もし，遅いメモリを使用する時はハードウェアの項で説明した様に，DSとDSACKをエッヂ信号として扱う工夫が必要になる。

同様の理由により，68020のシステムには従来の8ビット系の入出力ディバイスを接ぐべきでない。この8ビット系のチップの使用限界は，10MHzの68000／10であり，12.5MHzの68020でもトラブルの原因となる。

68020のシステムには，高速の16ビットに対応で68000系のI／Oディバイスか，これに相当するものを使用すべきである。

さらにできる事なら，I／Oの様な低速ディバイスは，別にI／O専用のMPUを持たせて（EX. 68008）そこと，デュアル・ポートRAM等を通じて入出力を行なう工夫が望ましい。

こう考えてくると，68020MPUを使用するシステムのイメージがある程度，明確になってくる。それは大きく二つに分けられ，それぞれのシステムの要求される特性に合わせられるものである。

比較的小規模で高速の処理のみを目標とするシステム

この様なシステムを称して，「瞬間最大風速マシン」という言い方をしているが，通常は，そよ風が吹いている位のMPUの能力しか使わず，そのほとんどはアイドル状況にあるが，ひとたび何らかの事態に対応して，もしくは遅延の許されない処理が重複する事がある様な事態となると，その高い処理能力を極限迄使う様な応用に用いる場合である。

この種の応用では，リアル・タイムの処理が当然重要であり，ある仕事を単位時間内に処理する能力が要求される。そして，比較的小規模であり，MMUは用いられず，FPCPも応用によって選択される。また組込みの装置等に用いられ，CPUのシステムとはとらえられないものが多い。それよりも，あるソフトウェアのループを1周する時間が限定されている様な場合に，それをどれだけの時間で行なえるかが最大の問題点となる。この様なシステムは，上位アドレスのデコード範囲を減らし，バス幅を32ビットと広くとり，キャッシュを使い，メモリも高速スタティックを使う。もしROM化されている部分のアクセス・タイムが遅ければ，これを高速RAMに移して実行する。ソフト的にも，PC相対，メモリ間接等のアドレス計算にMPUの内部サイクルのかかる複雑なモードは極力避け，レジスターレジスタ演算の様な高速演算を多用する事によって，ソフト的に高速化する。更に，この高速タイト・ループを実行している時，不要不急の割込みが入るのを避けるために，割込みのマスク・レベルを上げ，緊急割込み以外入らない様にする（もし，データが失なわれる心配があれば，別にプロセッサを付ける，FIFO

を付ける等，ハードで対応する)。

この場合，キャッシュ・メモリは始めから，イネーブルにしたままで問題はない。それは，MMUを持たないので，コンテクスト・スイッチを行なっても同一のアドレスが発生してくる心配がないからである。また高速スタティックのメモリを幅広く，必要充分の量を与える事により，デコード時間による性能劣化を極力避ける事を目標とする。

この様なシステム構成により，高速単機能のシステム構成を行なう事ができる。

こういった場合，データ・バス・バッファやアドレス・バス・バッファ，コントロールによる遅れを除くため，ボードを多数に分割してのバス・システムを作る事はほとんど行なわない，バス化できるのは入出力装置関連のみであり，主な仕事を受け持つCPU部分は，MPUの周辺近くに必要なRAM／ROM，コントロール等を全て配置し，信号の遅れを最小限とする様に構成する。つまり，外部と低速の通信を行なう部分以外は全てを包含するコンパクトなシステムを構成する必要がある。コンパクト化する事により先に問題点として述べたDSACKの遅れの様な問題の発生を避ける。もちろん，DSACKラインのバス化等も行なわない。

ミニコン的な使用法

これは，場合によりMMUを付けたり，FPCPを付けたりする。特にMMUの最高の能力を求めない場合には68030を用いる場合が多い。システムは，その融通性を高めるためVMEや，その他のバス・システムをとり，入出力の形態もシステム規模も変えられる様になっている。また仮想マシンの構成をとるものが多い。ソフトウェアも，本格的なOS，例えばUNIX等のマルチ・ユーザー・マルチ・タスクの環境をつくる。この様なシステムでは，OSによるタスクの切替時間が比較的大きく，システム全体の効率を向上させる事，ユーザー側のソフトウェアを容易に作成，実行させ得る事の方に重点が置かれている。そこで，コンパイラ用の複雑な遅いアドレスモードが多用される。

この様なシステムでは必然的に大規模のメモリが必要になり，物理アドレスの幅も必要とされるので68020の広いアドレス範囲を有効に利用しなければならない。この様な広いアドレス幅，大規模のメモリは当然，1メモリ・サイクルの時間を引き伸し，MPU本体の性能を低下させる。そこでこれを防ぐために，MPUの周辺にローカルのアドレス／データのキャッシュ・メモリを配置したりDMA用のデータ・ハイウェイを別付けする。こうしたシステムでは，本書内に概説した各種ハードウェア上の注意点を厳重に守るべきである。それは，MPUのシステム・バスに対する速度とバス上のディバイスの速度がしばしばミス・マッチを起こすからである。こういったシステムでは，その性能の向上に最も大きく貢献するのは，データ・バスの幅である。本来的な意味で32ビットのMPUの意義はこの点にある。汎用のソフトウェア，特に高級言語の扱う関数の演算長は長い。これをサポートするハードウェアは，そのメモリの幅と内部のALUの幅に，その主な性能を依存する。従ってこの様な応用には，32ビットMPUが適するのである。

新しい活用

今述べた二つの活用法以外に，この2例の特性を合わせ持つシステムが開発されている。これが最近広まってきているワークステーションである。これは，コンパクトにして，MPUの実行部は高速化しディスクや通常の入出力は従来のミニコン的な使い方をする。特徴としては，グラフィックのインターフェースをMPU部から直にドライブしている。そして，ミニコンのOSであるUNIXや，この系列のOSを使い，シングル・ユーザー，マルチ・タスクの環境を提供し，各ユーザーの仕事の特性に合わせたソフトウェアにより，個々の仕事は前者の例の場合の様に高速処理に重点を置き，その仕事のためのソフトウェアが大規模な点，データの量の大きな点はミニコンのアーキテクチャを用いて解決しているシステムである。

第10章
割込み処理

10-1　例外処理の概要

68000ファミリでは，一般にいう割込みを内部，外部の要因のものを総称して例外処理と言っている。この例外処理の共通点は，同じベクタ・テーブルを使い，ほぼ同様の手順でコンテクスト・スイッチが行なわれる事である。しかし，発生理由が異なれば当然対応も異なってくるわけで，この例外処理も幾つかのグループ分けができ，グループによりMPUの動作も多少異なるところがある(**表10-1-1**)。

10-1-1　外部割込み

外部割込みは，外部のハードウェアによるサービス・リクエストとして扱われ，ハード上１～７の優先レベルを持っている。ここで，７が最優先で６→５→４と優先度が下がる(**図10-1-1**)。

外部割込みは，内部のＳＲ内の割込みマスク・フ

表10-1-1　例外ベクタの割当て

ベクタ番号	ベクタ・オフセット 16進	ベクタ・オフセット 空間	割当て
0	000	SP	Reset : Initial Interrupt Stack Pointer
1	004	SP	Reset : Initial Program Counter
2	008	SD	Bus Error
3	00C	SD	Address Error
4	010	SD	Illegal Instruction
5	014	SD	Zero Divide
6	018	SD	CHK, CHK2 Instruction
7	01C	SD	cp TRAPcc, TRAPcc, TRAPV Instructions
8	020	SD	Privilege Violation
9	024	SD	Trace
10	028	SD	Line 1010 Emulator
11	02C	SD	Line 1111 Emulator
12	030	SD	(Unassigned, Reserved)
13	034	SD	Coprocessor Protocol Violation
14	038	SD	Format Error
15	03C	SD	Uninitialized Interrupt
16 Through 23	040 05C	SD SD	(Unassigned, Reserved)
24	060	SD	Spurious Interrupt
25	064	SD	Level 1 Interrupt Auto Vector
26	068	SD	Level 2 Interrupt Auto Vector
27	06C	SD	Level 3 Interrupt Auto Vector
28	070	SD	Level 4 Interrupt Auto Vector
29	074	SD	Level 5 Interrupt Auto Vector
30	078	SD	Level 6 Interrupt Auto Vector
31	07C	SD	Level 7 Interrupt Auto Vector
32 Through 47	080 0BC	SD SD	TRAP #0-15 Instruction Vectors
48	0C0	SD	FPCP Branch or Set on Unordered Condition
49	0C4	SD	FPCP Inexact Result
50	0C8	SD	FPCP Divide by Zero
51	0CC	SD	FPCP Underflow
52	0D0	SD	FPCP Operand Error
53	0D4	SD	FPCP Overflow
54	0D8	SD	FPCP Signaling NAN
55	0DC	SD	Unassigned, Reserved
56	0E0	SD	PMMU Configuration
57	0E4	SD	PMMU Illegal Operation
58	0E8	SD	PMMU Access Level Violation
59 Through 63	0EC 0FC	SD SD	Unassigned, Reserved
64 Through 255	100 3FC	SD SD	User Defined Vectors (192)

SP＝Supervisor Program Space　　SD＝Supervisor Data Space

ラグにより制御され，レベル7を除き現在のマスク・レベルを超えるレベルの割込みしか受付けない。また，あるレベルの割込みを受け付けると，割込みマスクはそのレベルにセットされ，「RTE」で元に戻るまでは同一レベルの割込みの受付を禁止する（もちろんプログラムにより割込みマスク・レベルを変更すれば受付ける(**図10-1-2**)。)

これに対し，全ての内部割込み及びバス・エラーは禁止できない。その代わり，リセット以外の全ての内部発生例外処理は，外部割込みのマスク・レベルの変更も行なわない。

例えば，バス・エラーの処理の始めであっても，外部割込みは受付けられる。

これは外部割込みは元々OSの資源であり，OSが制御すべきものであるため外部割込みの管理を放棄する事はできないからである。

外部割込みは，外部から割込みベクタ番号を送って処理の行き先を有するベクタをMPUに指示できる。この外部から送るベクタ番号は，実際には0〜255までの全てが使える。そこで割込みにより，ベクタ番号2のバス・エラーの処理を起動するような事も可能である。ユーザー専用の外部割込み用ベクタ・エリアがあるが，実質的には空きベクタ・エリアであり，将来にわたり空きエリアとなっていてユーザーに開放されていると思えば良い(**表10-1-1**)。

外部割込みは，更に前述したベクタ割込みとオート・ベクタ割込みに分けられる(**図10-1-3**)。

オート・ベクタ割込みは割込みアクノレッジ・サイクルで，ベクタ番号を引き渡す代りに単に68020なら「AVEC」。他のファミリでは「VPA」を与えるだけで割込みアクノレッジ・サイクルを終了させる事ができる。MPUはこの時，割込みの優先順位レベルにより特定のベクタ番号を参照する。これは例外ベクタのテーブル（**表10-1-1**）にLevel 1〜7, Interrupt, Auto Vectorと示されているベクタ番号である。

特定の外部割込みの優先順位がベクタ・テーブルと結びついているのはこの七つだけで，他のベクタは割込みの優先順位との相関関係はない。どの割込みのレベルも，任意のベクタ番号をMPUに与える事ができる。

10-1-2　内部割込み

内部割込みも幾つかの優先順位，及び機能レベルに分けられている。OSが動作している環境が想定され，OSの扱うべきレベルの差がはっきりと区別される様になっている。まず，内部発生の割込み，つまり例外処理の内，内部発生のものは大きく二つに分けられる(**表10-1-2**)。

一つは，システムのエラーに関するもの，もう一

図10-1-1　外部割込みの優先順位

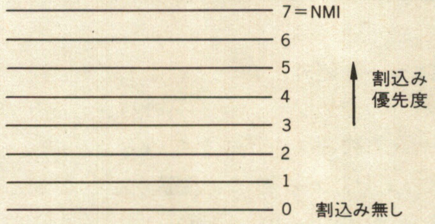


図10-1-2　外部割込みの禁止フラグ

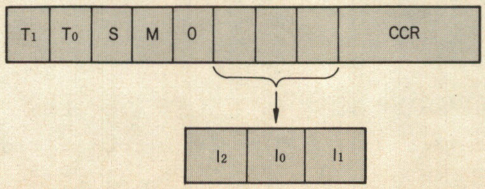


図10-1-3　68000ファミリの例外処理発生要因別ランク分類

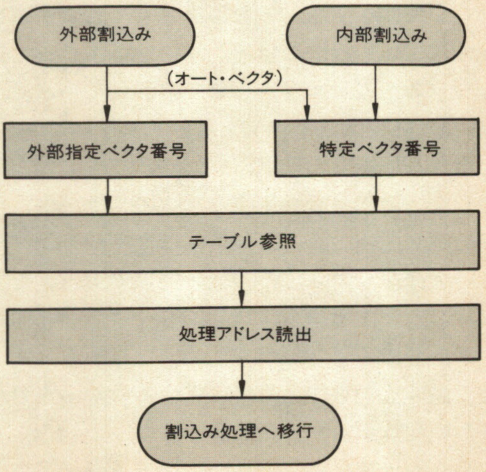


表10-1-2　68020 例外のグループ

グループ/優先順位	例外および相対的優先順位	特性
0	0.0—Reset	Aborts all processing (instruction or exception) and does not save old context.
1	1.0—Address Error 1.1—Bus Error	Suspends processing (instruction or exception) and saves internal context.
2	2.0—BKPT #n, CALLM, CHK, CHK2, cp Mid-Instruction, Cp Protocol Violation, cp TRAPcc, Divide-by-Zero, RTE, RTM, TRAP #n, TRAPV	Exception processing is part of instruction execution.
3	3.0—Illegal Instruction, Line A, Unimplemented, Line F, Privilege Violation, cp Pre-Instruction	Exception processing begins before instruction is executed.
4	4.0—cp Post-Instruction 4.1—Trace 4.2—Interrupt	Exception processing begins when current instruction or previous exception processing is completed.

0.0は優先順位最高，4.2は最低。

つは，ソフトウェア上の処理により，故意またはデータの値によって発生させられるものである。

システムのエラーによるものを更に分類すると，五つに分けられる(**表10-1-3**)。レベルAのリセットは別扱いとして，レベルBにアドレス・エラー，バス・エラーがある。

この二つのエラーの発生時には，MPUは内部状態をシステム・スタックの上に退避し，68010以上のMPUはエラーの修正後，継続実行し得る。

この内，バス・エラーは他の章でも触れているがメモリとのハンド・シェークの異常を示し，仮想マシンでは主記憶の再設定を要求する処理要求となる。アドレス・エラーは68020では命令ワードが奇数から読出されようとした事，つまりプログラム・カウンタPCが異常な値をとった事を示し，命令が順当に実行されている場合には起こり得ないエラーであり重大なシステム・エラーとして扱われている。

68020以外では，オペランドのメモリ・アクセスの時，奇数番地からワード，ロング・ワードのアクセスを行なった時にも発生する。このエラーは，バイトで処理していたタスク（例えば文字列を受け入れるという様な）から，その後の処理でデータをワードで扱おうとする場合等に起こりやすい。68000系のプログラムでアセンブラを使い，マルチ・タスクの環境で動くものを作る時に，打合わせが不充分だとたびたび起こるエラーである。

次のグループCは，何らかのソフトウェア上のミス，またはデータ処理上のミスによって発生したと考えられるもので，プログラム上のバグの発生，またはデータ入力のミスを示唆している。

表10-1-3　68000ファミリの例外処理発生要因別 ランク分類

A	Reset
B	Address Error Bus Error
C	Illegal Inst Zero divide Privilege Line Aエミュレータ Line Fエミュレータ
D	CHK, CHK2 TRAPcc CP TRAP Trace
E	Sprions Interset Format Error Co-Procei Pre Vio

68000系のMPUが一般に言う暴走を起こすとILLEGALまたはLINE−F, LINE−Aの例外処理によって止められるか，バス・エラーによって止められるかする。その時，システム・スタックが正常ならシステムを自分である程度回復処置をする事ができる。

ILLEGALは実行コードの空きを示し，Line−F, Line−Aも実行コードのない事を示している。同じ実行コードがないといっても，Line−FとLine−A

は将来の実行コード用の空きエリアで，現にLine－Fの一部は68020ではコ・プロセッサ用に使われている。したがって，ILLEGALは，Line－FとLine－Aは含まない。命令にある「ILLEGAL」は68000ファミリの将来における拡張にも実行コードのない事を保証したものである。

グループＤは，ソフトウェアにより故意に発生させられる例外処理である。この系統は，通常ＯＳへのサービス要求である単なるソフトウェア割込み＝「TRAP」や，データの値を調べ，結果によりソフトウェア割込みを発生する「CHK, CHK2」，コンディション・コードの内容によって割込みを発生する「TRAPcc, TRAPV, cpTRAPcc」がある。

この中にトレースも含めている。これは68000ファミリの１ステップ実行，分岐までの実行（68020）は，ソフトウェアのみでサポートでき，スタック・フレーム内のＳＲ相当部のメモリ操作と，トレース・ベクタによって示されるソフトウェアの処理により実現できるためである。

グループＥは付属回路，素子の異常動作に関するもので，システム構成により各々の処理の内容は一概に議論できない。ただし，バス・エラーの様にＭＰＵの内部状態の全ての内容のセーブが行なわれないため，再実行はいずれにせよ不可能である。一般にこれらに対する処置は，システムの初期化のやり直し，外部ディバイスの再設定等の手段をとる。

この他68020では，68881：フローティング・ポイント計算用コ・プロセッサ，68851：ページ・モードMMU，略して，68881をFPCP，68851をPMMUと言っているが，これらによる例外処理も上記のＥ分類の発生原因と同じ要領で定義されている。

この点に関してはユーザーズ・マニュアルのコ・プロセッサの章（第８章・11章）を参照されたい。コ・プロセッサはＭＰＵとの専用通信手順により，様々な例外処理を起動し，コ・プロセッサの使用上のミスも例外処理を引き起こす。

10-1-3　代表的な内部割込み発生条件概説

2→バス・エラー：外部回路によるＡＳの時間幅の上限違反，又は，MMUからの信号

図10-1-4　68000ファミリの割込みの受付手順

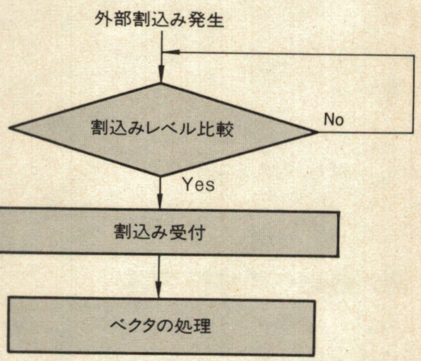
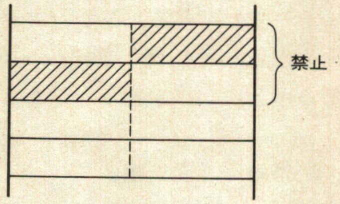


図10-1-5　68000ファミリの命令ワードのアドレス・エラー



3→アドレス・エラー：プログラム・コードのワード整列違反，又は，68000以下でのワード非整合アクセス
4→不当命令：命令コードが正しくない
5→０分母：割算命令の分母が０
6→CHK, CHK2命令
7→TRAPV, cpTRAPcc命令
8→ＯＳにのみ許される特権命令のユーザー使用
9→トレース
10→ＯＰコード＄Ａ命令（命令の空きエリア）
11→ＯＰコード＄Ｆ（コ・プロセッサ命令）
12→未定義
13→コ・プロセッサとの通信法違反
14→フォーマット・エラー：スタック・フォーマット・モジュールのフォーマットのエラー，コ・プロセッサのリストア時にも発生する。
15→68000ファミリのI／Oで割込みベクタがMPUによって未設定の場合
16～23→未定義
24→割込みアクノレッジ・サイクルでのバスエラー
24～31→オート・ベクタ　レベル１～７
32～47→TRAP　０～15のベクタ

48〜54→FPCPの定義による
55→未定義
56〜58→PMMUの定義による
59〜63→未定義
64〜255→ユーザー外部割込み用
（先頭の番号はベクタ番号）

10-2　例外処理への移行手順

例外処理は，発生→受付け→ベクタ読込み→ベクタ・テーブル・アクセス→処理の手順で進行する。そこで，受付け，ベクタを中心に説明する。これはベクタ・テーブルを読み，そこのアドレスから処理を行なう事は広く行なわれているので，ファミリの特徴のある所を中心にするためである。

10-2-1　例外処理の優先順位

例外処理は，そのシステムにおける重要度に応じて何種類かにグループ分けされている。区分理由としては，MPUがその受付けた例外処理にどう対応するか，その処理がどういう意味かにより変化する。

〔68000／08の場合〕

グループは3種類に分類され，0，1，2の優先順位が付けられている（**表10-2-1**）。

〈グループ0〉

グループ0は，その事象が発生したらMPUの状況に関わりなく，即時に例外処理へ移行するもののグループである。また，このMPUタイプではBusError，Address Errorがここに含まれる。これはこのタイプのプロセッサでは，実行を中断した命令を再実行する事ができないからである。この時，Bus Error, Address Error時，スタックは8ワードが使われ，若干のエラー情報が含まれる。

〈グループ1〉

このグループは実行中の命令が終了し，次の命令がプリ・フェッチされている時に例外処理へ移行するグループで外部ハードウェア割込みが含まれる。スタックは3ワードのみで，PCとSRがセーブされる。

〈グループ2〉

表10-2-1　例外処理移行のタイミングと優先順序

Grop		
0	Reset, Bus Error Address Error	即時
1	Trace, Illegal Priv	命令の切れ目
2	TRAD, TRAPV CHK, Zero Div	命令

このグループは通常のソフトウェアとして，命令の流れに乗って例外処理へ移行する。ソフトウェア割込みであるTRAP命令がこのグループに入る。スタックはグループ1と同じで，PCとSRのみセーブされる。

〔68010／12の場合〕

68000／08の場合のグループ0から，Bus Error, Address Errorがはずされて，グループ0はRESETのみとなり，グループ1にBus Error,Address Errorとなり，グループ2，3は68000のグループ1，2がそのまま移行する（**表10-1-2**）。

これはRESETはMPUの状況を完全に初期化し回復を不能にしていいるがBus Error, Address Errorに対しては，MPUの内部状況をシステム・スタック上に退避して再実行を可能にしている。これは仮想マシンを可能にするために付けられた新しい機構で，実行中の命令は保留状態（Suspend：サスペンド）になっていると言っている。このBus Error, Address Errorでは，命令の再実行を可能にするため，29ワードがスタックされる。

〔68020／30の場合〕

このグループは，68010／12の場合とグループ0，1は同じだが，2，3は全体が再構成されて**表10-1-2**の様に分けられている。これは分類の理由付けがグループ2は命令の実行の一部として，グループ3は命令が実行される前のチェックによって，グループ4は現在の，またはその命令が実行された結果としてという様に仕分けし直され，優先順位も付け直されたからである。この分類についてはこの章の「ソフトウェア割込み」を参照されたい。

当然の事であるが，これらのグループ分けされている順序は優先順位を示し，同時に複数の例外処理が発生した時は，優先順位の高いものから実行され

る。また重なった場合，これらのグループの特性は保たれる。例えばTRAP命令と外部割込みが重なった時は，先にTRAPの例外処理が実行され，新しい実行アドレスがＰＣにセットされ，最初の命令のプリ・フェッチ後に外部割込み受付の動作を行なう。

注意すべき点は，Bus Error,Address Errorの様にメモリのバス・サイクルによって発生する例外処理では，命令プリ・フェッチのために，実行中の命令と直接は関係のないバス・サイクルが行なわれそこでエラーが発生するため，ＭＰＵは現在，実行中でない命令によって実行が中断する事があり得る。そこで，バス・エラーの解析プログラムではこの点に充分注意すると共に，メモリの境界附近にプログラムを置く時は，プリ・フェッチにより余分のバス・エラーを起こさない様に注意する必要がある。

ただし，68020／30では，命令のプリ・フェッチのみでは例外処理は起こらず，命令のデコーダに入ってから起きる場合もある。

10-2-2　ベクタ・エリア

例外処理は各々の発生原因と対応したベクタ・エリアを持ち，このベクタ・エリアにその各々の発生原因に対応する処理プログラムのスタート番地を入れておく事により，例外処理を要求する事態が発生すれば，ＭＰＵは現在の処理を中断して，発生原因により指定されるベクタ・エリアから該当処理アドレスを読み出し(フェッチ)，処理の内容により分類されるグループごとに決められたＭＰＵの内部情報をスタック・フレームとしてシステム・スタックまたは割込みスタック(68020／30)の上に記入する。

その後，更に上位の例外処理の要求がなければ，先にフェッチしておいた実行アドレスから処理を始める。この時，ＳＲ（ステータス・レジスタ）内のＳフラグ（スーパーバイザ・ステート・フラグ）は自動的にセットされ，スタックもシステム・スタックか割込みスタックに切替えられている。

ベクタ・エリアは68000ファミリのどれも１Ｋバイトのメモリ空間に256ロング・ワードのベクタを持ち，アドレスの低い方から順に０～255の番号を付けこれをベクタ番号と呼ぶ。

68000／08では，このベクタ・エリアは０番地から＄３ＦＣまでの１Ｋバイトに固定されている。

68010／12／20／30では，ＶＢＲ(ベクタ・ベース・レジスタ）と呼ばれる32ビット長のレジスタが，このベクタ・エリアのアドレス空間中の位置を指示する。このＶＢＲは，リセット時には０がセットされ68000／08と互換がとれ，回路上のリセットが行なわれて，リセット・スタート時の，ベクタの読出し時には，０番地よりのベクタ・エリアが有効となる様になっている。

このＶＢＲを用いた時の各ベクタのアドレスは，

ＥＡ＝ベクタ番号＊４＋（ＶＢＲ）

で与えられる。もちろん，この時ＶＢＲの内容はワード整合を必要とする（偶数でなければならない)。

10-2-3　ＶＢＲの効用

このＶＢＲによって，システムは複数の例外処理用のベクタ・エリアを持つ事ができる。一般にこのベクタ・エリアは，ＯＳの機能のスタート，外部，内部からの割込み処理のルーチンへの転送に使われるので，このベクタが複数あるという事は，一つのシステム・ハードウェア上に複数のＯＳが存在できる事を意味する。

図10-2-1の様に，あるシステム・ハードウェアが存在し，その上にまずMASTER－OSがあり，さらにその上に，Ａ，Ｂ，ＣのサブＯＳが存在したとする。そのサブＯＳの上に様々なユーザー・タスクが動いている。もちろん，MASTAR－OS上にも幾つかのユーザー・タスクが動いているとする。

図10-2-1　マルチOSシステムの場合のVBR，及びシステム全般との関係

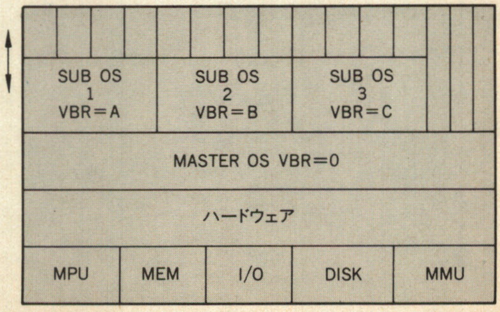


図10-2-2　システムコールがサブOSからMASTER－OSまで行ってサポート

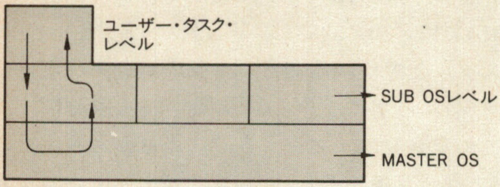
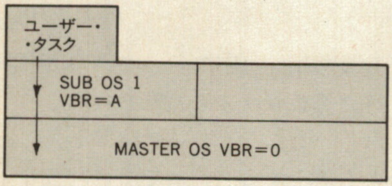


図10-2-3　ユーザー・タスクからMASTER－OSへシステム・エラー発生を知らせる



各ユーザー・タスクからはプログラムの実行に伴い，種々のシステム・コール（ＯＳコール：ＯＳに対して何らかの仕事，サービスを要求する事）が各々のＯＳに対して発生する。もちろん，システム・コールの種類や形態はＯＳの種類によって異なるが内部のソフトウェア割込み（TRAP，ＣＨＫ他）によって各ユーザー・タスクからそのＯＳへの移行がなされる。そこで，この様なソフトウェア割込みに対応するベクタは，各ＯＳ専用のものをMASTER－OSの管理下にMASTER－OS側に作成し，そこで各ＯＳのシステム・コールをサポートし，さらにシステム全体に関わるトラブル，システムのハードウェア・サポートを必要とするシステム・コールに対しては，０番地にあるのと同じMASTER－OSのベクタのコピーを使う事により，MASTER－OSの直接のサポートを要求できる。または，サブＯＳが自己の内部でサポートしきれない事柄に関しては，このMASTER－OSのコピーされているベクタ部分のMASTERとＳＵＢのＯＳ間の行き来として決められているソフトウェア割込みの利用により，MASTER－OSのサポートを受ける事ができる。

サブＯＳ上のユーザー・タスクにしてみれば，自己のタスクは自己のＯＳのサービスを受けているごとくに見える。

ユーザー側が標準のＯＳの何種かをヒューマン・インターフェースとして同時に動かすことができるシステムを要求する場合に，ＶＢＲによりベクタ・エリアを複数持てることによってこの様なシステムを可能とする。

また，注意する点は，ここでＭＰＵがスーパーバイザ・ステートで動いているのは，MASTER－OSの部分だけで，各サブＯＳはユーザー・タスクをサポートする様になっており，ユーザーごとのスーパーバイザ・コールが専用のベクタから，各々の処理を行なうサブ・ＯＳであるが，ユーザー・モードで動いている部分に渡される様に考えられている。

この時，スーパーバイザしか使用を許可されていない命令をサブＯＳが実行すると，プリビレッジ・バイオレーションの例外処理が起き，この場合，そのベクタはMASTER－OSのコピーで，MASTER－OS内を指しMASTER－OSがその正当性をチェックして，相応の処置をとり，サブＯＳに実行を戻す。このマルチＯＳに関しては第４章のマルチＯＳの項を参照されたい。

10-3　例外処理とバス・エラー

このバス・エラーは，命令の読取り，データのリード／ライト，割込みベクタ番号の引渡し等のメモリ・バス・サイクルに際して，ＭＰＵがバスを使っている信号であるＡＳ（アドレス・ストローブ：Address Strobe）が出力されている時間を測るタイマをＭＰＵの外付回路として付け，ＡＳの時間がそれぞれのシステムによって決められた時間内に入っているか否かをチェックさせ，この規定時間に違反した場合，また時にはメモリのパリティ・エラーの信号を加えて，メモリ・サイクルのエラーとしてバス・エラーの信号入力に与える事により起動される例外処理である。

メモリのパリティ・エラーの場合は別として，ＡＳの信号の時間が規定外となるのは，ＭＰＵがアクセスしているアドレスにアクノレッジを返してくる回路がない，つまり，そこにはメモリがない事を意味すると考えられる。この様なアクセスが起きる原因は幾つか考えられるが，その代表を以下に示す。

1)．アクセスを行なった部分に情報，データ，プログラム等は存在するが，それはその時点で外部記憶のディスクの中に存在していて，主記憶中には存在していない(仮想マシンのアクセス・フォールト)。
2)．物理的には存在するメモリ部分にアクセスを行なったが，MMUのその部分のセグメントが与えられていない。
3)．スタックやヒープにオーバー・フローが生じ，指定された範囲外になった。
4)．プログラムのミスによって生じた（正しくないアクセスの結果)。

バス・エラーの処理は，こういった原因を分析して回復させる事のできる場合は回復させ，無理な場合はそのタスクを中断させるといった処置をとる。

このための基礎となる判断材料は，システム・スタックの中にエラー情報としてセーブされている。システムの回復のできない68000／08で７ワード，回復処置をとれる68010／12では29ワード，68020／30では命令の実行時の状態によって16ワードの時と46ワードの時がある。68020／30ではFormat Codeによりスタックの性質，長さが区別され，他の普通の例外処理，コ・プロセッサの場合のスタックの処理等と分けられている。

この長いスタックに入っている内容は，通常の割込みの時のＰＣとＳＲだけでなく，アクセス・アドレス，R/W，ＦＣ，更に68010以上ではインストラクション・パイプの内容や，内部のレジスタの内容まで含まれる。

この様子はちょうどＭＰＵ自体を一つのコンピュータのシステムに見立て，バス・エラーの入力はシステムに対する割込みと考えると良く一致する。割込みにより，コンピュータ・システムが割込みを受けると自分の内部状況を一時システムの外へ退避させ新たな実行を始める。この様子は，68020／30／10／12のバス・エラーの処理とほぼ同一である。これは，ＭＰＵが内部マイクロ・プログラミングでできていて外部からの命令を内部では幾つかの細かいサイクルに分けて実行している事から生ずるもので，普通のシステムの命令実行がこの内部マイクロ・プログラミングの実行になぞれる訳である。したがって，バス・エラー，アドレス・エラーはこのマイクロ・プログラムによるソフトウェアに対しての割込みと見る事ができ，マイクロ・プログラムの実行部が割込みを受けて，自分の内容を外部にセーブして新たなプログラムの実行にかかる事になる。

10-4 バス・エラーの入力に対する ＭＰＵの例外処理以外の対応

バス・エラーの入力ピンには，先に説明した様にＡＳ信号の長さが，システムによる規定値に違反した時に信号を与え，アクノレッジであるDTACK，ＶＰＡまたは68020／30のDSACK０／１の代りをさせかつエラーがあった事をＭＰＵに知らせるが，この信号に対するＭＰＵの対応は「例外処理のバス・エラー」のみではない。これ以外に他の信号や，その時のメモリ・サイクルの内容によって別の対応が生ずる。以下にこれを概説する。

1)．先に説明した「例外処理のバス・エラー」を起す。この時は通常のメモリ・サイクルであり，アクノレッジ信号より先，もしくはほぼ同時にバス・エラーの入力に信号が与えられた時に生ずる。
2)．外部割込み要求に対して，ＭＰＵが割込みアクノレッジ・サイクルを起こし，その中で割込みベクタ番号フェッチのためのバス・サイクルを起こしている時，アクノレッジ信号もＶＰＡも与えられずにバス・エラーの信号をＭＰＵに与えた場合。→スプリアス割込み（例外処理中のベクタ番号24番）となり，割込みを受けたのに，割込みアクノレッジに対して呼応してくるものがいなかった事を示す例外処理を走らせる事になる。
3)．バス・エラーの信号と同時にHALT信号を与え，そのバス・サイクルを一時終らせ，次にバス・エラーの信号を消し，その次にHALTの信号を消す。→このバス・サイクルを再度，始めから繰返す。例外処理は起こらない（この処理はRe－RUNと呼ばれている)。
4)．アドレス・エラー，バス・エラーの例外処理が起きて，ＭＰＵの内部状況をシステム・スタック上に展開するためのバス・サイクルでバス・エラーが

発生した場合。→MPU・HALT

この場合はダブル・バス・フォールトと呼ばれている処理で，ＭＰＵのシステム・スタックに異状が生じ，ＯＳが，もはやシステムを正常に運営できない事を意味する。

リセット直後のＰＣ，ＳＰの初期値を読込んでいる時も同じくHALTとなる。

システム・スタックが正常でないというのはＯＳ内でサブ・ルーチンも割込みの処理もできない事を意味するため，システムにとって最も基本的な部分が異状になった事を意味している。そこで，この場合にはＭＰＵはHALTの信号線を内部からドライブ出力し，ＭＰＵの全ての動作を停止し，その時のメモリの内容等に変化を起こさせない様にする。

そして，RESETが入力されるまでその状況を維持する。ＭＰＵがダブル・バス・フォールトに入った事は，HALTの信号線が外部理由がないのにドライブされる事ではっきりと区別でき，外部からの介入を待つ事になる。したがって，HALTの信号線は外部からアクティブＨにドライブしてはいけない。オープン・コレクタのドライバを使用（第９章15及び図9-14-3参照）する。

5)．68020／30の場合のみ

ブレーク・ポイント命令の実行において，プログラムのフローをイリーガル命令発生の例外処理へ移して，中断する方向へ持っていく。

このブレーク・ポイント命令が実行されると，ＭＰＵのファンクション・コードはＣＰＵスペースのコードとなり，アドレス・バス上にブレーク・ポイント・コードとその指定番号を出力し，データのリード・サイズはワードを指示してくる。そして，ＡＳとＤＳが出力されてくる。

これに対して命令を継続的に実行させたいという状況の時は，データ・バス（D31～D16またはD31～D24に２回）にブレーク・ポイント命令に換わる命令のＯＰコードを乗せて所定のアクノレッジを返してやる。すると，ＭＰＵはこの命令を命令デコーダのパイプに入れて実行し，プログラムのフローは先に進む。

中断させようとする時は，DSACKxの代りにバス・エラーをＭＰＵに与える。データ・バスは不定で構わない。バス・エラーを検出した68020／30は，内部でイリーガル・インストラクション（不法命令発生）例外処理を起動し，そこでプログラム・フローを中断しＯＳ側に戻す。これはハードウェアの補助により，デバッグを行なうための機構である。

6)．コ・プロセッサとの通信のためのコ・プロセッサ専用空間へのアクセス時のバス・エラー→ライン＄Ｆエミュレータの例外処理が起こる。これによりプログラムはFPCP等のコ・プロセッサが付けられているか，否かをチェックする事ができる。

10-5 アドレス・エラー

アドレス・エラーはバス・エラーと異なり，ＭＰＵの内部で発生する内部割込みで68000／08／10／12では，ワード，ロング・ワードのバス・サイクルの時に実行アドレスが奇数の場合，つまり一つのワードを二つのメモリ・ワードにわたってアクセスしようとした時に内部チェックにより実行前に例外処理へ移行させられる。この場合はそのメモリ・アクセスが命令であろうとデータであろうと同じ扱いで発生する。

68020／30の場合は，メモリ・ワードにまたがってのアクセスは許されているため，このデータによるアドレス・エラーは起こらない。この場合には，唯一の制限「各命令及びベクタはワード整合を必要とする」との条件に反した場合にのみ発生する。

しかし，68020／30に限らず命令のコード自体はコンパイラで作られたものもアセンブラで作られたものも命令のコードがワード整合に合致しているか否かはコンパイル，アセンブルの時点で一般にはチェックされているのが普通である。実際に命令のフェッチ時にアドレス・エラーが発生したとすれば，それは即ち正常な実行状態にない事を示している。

これに較べると，データにおけるアドレス・エラーはアセンブラで記述した場合には実行時エラーとしては良く起こる。これはバイト長のデータ，つまり文字列の様なものを扱っていると，ポインタが奇数になるのは，しばしば起きる事で，そこにワード

のデータを追加しようとした場合などに確率½で発生する。

10-6　特権命令違反

プリビレッジ・バイオレーション。これは直訳すると特権違反となるが，図10-6-1にこの特権違反の命令を示す。この特権とは何の特権かという事になるが，これはＯＳが有する特権，つまりＯＳはシステム全体のコントロールをする特別な部分であり，この特権に対しての違反が起きた場合に起きる例外処理である。もう少し具体的に表現すると，ＯＳはＳＲ内のトレース機能のコントロール，割込みのマスク・レベルの制御，スーパーバイザ・フラグの制御等ＳＲ内の上位バイトの変更，修正を行なう機能，外部素子を制御するためのリセット機能，割込み待ちのSTOP機能，スーパーバイザ・モードにいながらユーザー・スタックに読み書きできる機能，同じ様にスーパーバイザ・モードにいながらユーザー・モードに割当てられているメモリをアクセスする機能，コ・プロセッサの内部状態を退避したり元へ戻したりする機能である。

これらの機能はＯＳが操作するものと考えられているので，これに反してユーザー・モードでこれらの命令を実行しようとすると例外処理プリビレッジ・バイオレーションが起動される事になる。

もっともマルチＯＳの環境下では，サブＯＳの方では自分はＯＳだと思っているので，これらの特権違反は繰返し発生する。そこでマスタＯＳはこの例外処理に対しては各ＯＳは登録要求が何なのか，割込みのマスクなのか，そのＯＳ上のあるプログラムのトレースなのかを判断して，登録し疑似的にマスタＯＳの内からサポートしてやらなければならない。

図10-6-1　MPUタイプによる特権命令の違い（MOVE from SRは68000／08では特権命令ではない）

68000/08	68010/12	68020/30
ナシ	ナシ	CP Restore CP SAVE
	MOVE TO SR MOVEC MOVES	
	Reset RTE STOP ORI to SR MOVE USP AND AND I to SR EOR I to SR MOVE to SR	

10-7　外部割込み手順の詳細

68000ファミリの割込み手順は従来の８ビット系のものと較べると若干複雑で，特にハードウェアの設計には，68000の特徴を良く理解しておく必要がある。

まず，68000の割込み処理はＭＰＵに受付けられると，ＭＰＵはベクタ・テーブル内の処理をベクタを参照して，各割込み処理のスタート番地を決定する。このベクタ・テーブルには255個のベクタが入っており（１ベクタは１ロング・ワード），その内の192ベクタが外部割込み用，25ベクタが内部発生の処理用，２ベクタがアクセス異常に対するものとなっており68020／30では更に11ベクタがコ・プロセッサ発生のベクタとして追加されている。

ＭＰＵは，RESET－START時は必ずこのベクタ・テーブルを初期化しなければならない。

68000での割込みは例外処理（EXCEPTION PROCESSING）と呼ばれる処理で，これは内部，外部割込み，アクセス異常等，あるプログラムの実行が中断され得る全ての場合を包含しているうちの外部発生の場合と定義される。

外部の入出力回路，タイマ等からの割込みにも，更にベクタード割込みとノン・ベクタード割込みに分けられ，更に１～７までの優先順位をハードウェアにより持たせる事ができる。

10-7-1　ベクタード／ノン・ベクタード割込み

割込みをＭＰＵに対して要求した側のハードウェアの内容によってこの２種に分けられる。

ベクタード割込みは，割込みを要求した側が自己

の割込みの要求に対してのアクノレッジ・サイクルの時に割込み要求側からベクタ・テーブル内の255の処理ベクタの内，どれを使うのかを1バイトのデータ（＝ベクタ番号）として渡して，ベクタ・テーブルのポインティングを行なう方法である。

10-7-2　ノン・ベクタード割込み（オート・ベクタ）

ノン・ベクタード割込みは，同様に割込みアクノレッジ・サイクルの時に，この割込み要求はベクタ番号を発生できない事を示す信号，68000／08／10／12ではＶＰＡ，68020／30ではAVECをＭＰＵに与えて，ベクタ・テーブル内の特定のベクタ番号による処理を要求するものである。この場合は，オート・ベクタと呼ばれ，専用のベクタ・エリアが七つあり各割込み優先順位と対応している。

10-7-3　割込みの優先順位

68000ファミリは割込み要求受付入力信号としてIPL2, IPL1, IPL0がプライオリティ・エンコーダにより（74ＬＳ148等）コード化された信号としてＭＰＵに与えられる。

このＩＰＬ２～０に与えられるコードを０～７と数値化して考えられるのが，割込み要求レベルとなり，レベル０は割込み要求のない状態，レベル７はノン・マスカブル割込み（最上位の）となる（実際のＭＰＵでは，この信号は負論理の信号であるのでＩＰＬ２～０が全てＬレベルの時がレベル７割込みを示している）。

図10-7-1　割込みマスク6，7と割込み要求6，7の場合の関係

MASKレベル＝7
要求レベル＝7　→割込み起こらず
MASKレベル＝7
要求レベル＝6　→割込み起こらず
MASKレベル7
要求レベル7　→6→7の変化でレベル7割込み起きる
MASKレベル7
要求レベル7　→割込み起こらず
MASKレベル6
要求レベル7　→レベルとしてレベル7割込み受付ける

このＩＰＬ２～０から与えられる外部の割込み要求レベルと，ＭＰＵ内のＳＲ内の割込みMASKレベル（I2～0）とを比較して，外部要求レベルが内部のMASKレベルを上まわっていれば（同じでは不可），その割込み要求は受付けられる。一度，ある割込みが受付けられると，割込みMASKは自動的に受付けたのと同じレベルにセットされ，次の同一レベルの割込みを禁止する。

この例外はレベル７のノン・マスカブル割込みでレベルが１～６までの割込みはレベル受付けとなっているが，このレベル７のみエッジ受付けとなっている。

レベル１～６の割込みは，同一レベル上に複数の割込み要求源があり，あるレベルの一つの割込み処理を行なっている間に同じレベルの他の割込みが発生し，先の処理が終了して，MASKレベルが元に戻るか，ソフトウェアで強制的に割込みMASKのレベルを下げるかすれば，次の割込みは，割込み要求ラインＩＰＬのレベルが複数の割込み要求のために，まったく変化がなくても受付けられる。ただしレベル７のみはノン・マスカブル＝禁止できないという性質上，特別扱いで割込みMASKレベルが７（つまり全て禁止）であっても，外部要求のＩＰＬレベルが一度どれかの別のレベルに変化し，またレベル７になった場合には割込みは起動される。もちろんMASKレベルが下がった場合でＩＰＬへの外部要求レベルが７の場合も受付けられる。

したがって，ノン・マスカブル割込みを要求するハードウェアはその要求内容にもよるが，新たにＭＰＵに処理を要求する時は，一度別のレベルにしてから新たに与えてやる必要がある。

割込みMASKのレベルをもう少し説明すると，マスク・レベル６の場合を考えてみるとレベル７の割込み（＝ノン・マスカブル割込み）しか割込みは受付けられない。

つまり新たなレベル７割込みが起こらない限り割込み処理が起きないという点では，レベル７のMASKレベルと同一である。

ただしレベル６の割込み源が複数あって，一つの

処理が終わった時点でＳＲのマスクレベルが６未満となった時，ＩＰＬのレベルが６を維持していても次の同一レベルの割込みを受付ける点では若干，差が出る(**図10-7-1**)。

10-8　外部割込み処理回路

68000ファミリでは,割込み要求に要するハードウェアの内容は同一である。

10-8-1　ＭＰＵ周辺

ＭＰＵの周辺には，割込みを受付けるためのエンコーダ，8 to 3のプライオリティ・エンコーダが必要で，これによりＭＰＵのＩＰＬ0，1，2に割込み要求信号を与える(**図10-8-1**)。

これと共に，外部の割込み要求ディバイスに割込みアクノレッジ・サイクルである事を示すIACKの信号を外部に出力してやる必要がある。これはファンクション・コード（ＦＣ０，１，２）をデコードして，３本ともＨレベルであれば68000／08／10／12ならそのままIACKとして，68020／30ではＡＳ出力後A16～A19の４本がＨであることとＡＮＤ条件でIACKを作り，割込みアクノレッジ信号とする(IACK信号)。通常のIACK信号は，ディジー・チェインを構成して，同一レベル割込み要求に対し，優先順位を設定する。また，このIACK時にアドレス線A1，A2，A3は受付けた割込みレベルを示す(**図10-8-2，3，4**)。多くのシステムで，このIACK信号とA1，A2，A3を合成して割込み要求の７レベルに対応した７レベルのアクノレッジ・サイクルを作っている。

図10-8-1　68000ファミリに必要な外部割込みレベルのエンコーダ

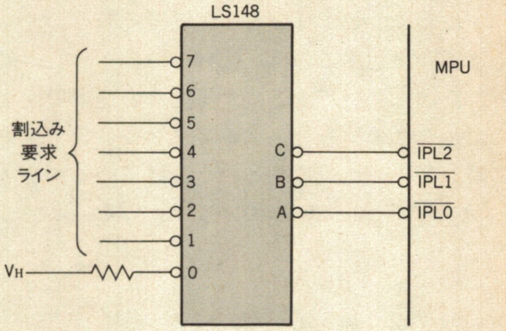


図10-8-2　68000／08／10／12における割込みアクノレッジ・サイクルのデコード例

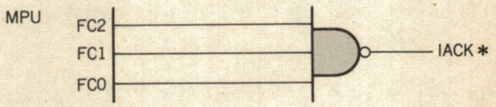


図10-8-3　68020の割込みアクノレッジ・サイクルのデコード例

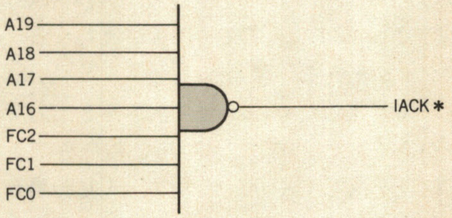
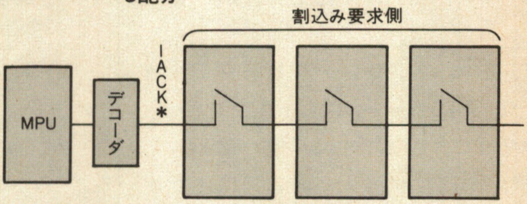


図10-8-4　割込みアクノレッジのディジー・チェインによる配分



MPUに近い側が高い優先順位を持つ。

10-8-2　割込み要求側

これに対して割込み素子側では，ＭＰＵのIACKサイクルに対して次の様な回路が必要である。

1).割込みを発生するためのドライブ回路を設ける。通常オープン・コレクタにて，ワイヤードＯＲとなっている割込要求ラインをドライブする。

2).IACKサイクルに入って，ＡＳが出力されてきた時，Ａ１，Ａ２，Ａ３にＭＰＵが受付けた割込みのプライオリティ・レベルを出力してくるため,これを自己の出力しているレベルと比較し，一致していればこのIACKサイクルに反応する。ここで反応できるまで自己の割込み要求は保持をする。この時，自分以下のレベルにIACKのディジー・チェインが行かない様にする。そのため，要求側でのデコードのみでなく,IACKでストローブされる必要がある。こ

れは，各割込みレベルに要求源が１個のみなら不用であるが，複数ある時は複数の要求源が応答しない様にする為に，必ず行なうべき処置である。

もう一つ注意するのは，**図10-8-1**のＩＰＬの信号は外部要因によって割込みサイクルの途中で変化するので，ここの信号を割込みアクノレッジ・サイクルの時，どのレベルの割込みを受けているかを示すA1，A2，A3との比較に用いてはいけないという事である。

3)．ベクタード割込みを使う場合

ベクタ番号発生用レジスタ，出力バッファ及びストローブに対してのアクノレッジを返すための部分をイネーブルする。ＤＳまたはＬＤＳが出力されてきたら，要求ベクタ番号を所定のデータ・バス上に乗せ，DTACKまたはDSACKxを返してやる。68020／30の場合，そのベクタ番号を乗せてやるデータ・バスの幅と指定位置によって，DSACKxのｘは異なる(ダイナミック・バス・サイジング参照)。

68000／10／08では，D7～Ｄ０が使われる。

4)．ベクタが出せない場合

回路の簡素化のために，ベクタ番号を出力できない場合がある。この時は，68020ならＤＳがきたらAVEC信号を返してやり，他のファミリの場合，ＬＤＳがきたらＶＰＡを返してやる。こうするとベクタ番号25～31のオート・ベクタがプライオリティ・レベル１～７に対応して使われる(オートベクタ)。

10-8-3　アドレス・デコーダの注意

ＣＰＵスペースの時または割込みのアクノレッジ・サイクルの時は，他の使われていないアドレス線が全部Ｈレベルにある事に注意がいる。これは通常68000ファミリでは，Ｉ／Ｏのアドレスを負のベース・ページ，つまりアドレスの最高位付近に持ってくる事が多く，この時，ファンクション・コードもスーパーバイザのデータとしてデコードされている場合が多いため，アドレスの上位８または16ビットが全てＨであり，かつＦＣ２，ＦＣ０ともにＨで，これはＣＰＵスペースのIACKサイクルとほぼ同一の条件である。したがって，Ｉ／Ｏ用のデコーダを作る時には割込みアクノレッジ・サイクルでＩ／Ｏにアクセスが入らない様に気を付ける必要がある(**図10-8-6**)。

図10-8-5　割込みベクタ・発生回路の例(レベル４)

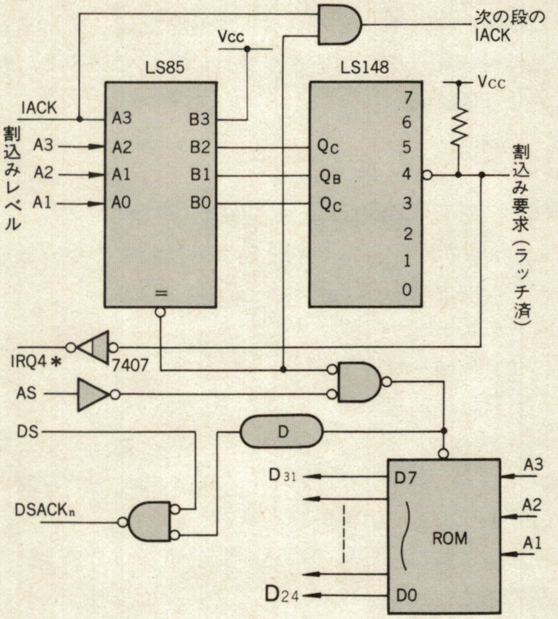
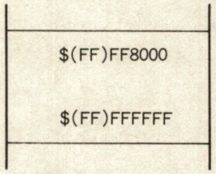


図10-8-6　マイナスのベース・ページ・エリア(Ｉ／Ｏアドレスとして多用する)



10-9　例外処理時のスタック・フレーム

68020では，仮想マシンのサポート，コ・プロセッサの命令中断に対するサポートのために，ＭＰＵ自体の命令を中断し，また回復しての継続実行ができる。

この機能は，ＭＰＵの内部の状態をシステム・スタック上（この場合ＩＳＰ）に展開し，これをまたＭＰＵ内に再設定する機能である。

このためＭＰＵの例外処理へ移行する状況によって，このスタックに退避すべきデータの量が大幅に

変わる。

通常の外部割込み，TRAP命令によるソフトウェア割込みの時はPC, SR，フレーム・コード・ベクタを退避し，これは４ワードとなる(**図10-9-1**)。

68000の他のファミリでは，この時PC, SRのみの３ワードの退避となっている。

この場合をフォーマット０としている。

これとほとんど同じものが，外部割込みが発生した時に作られる。これはフォーマット１としている。

命令実行後の例外発生によるスタックは６ワードあり，例外発生時のアドレスはその命令のアドレスであり，ＰＣ値は次の命令のアドレスとなっておりＲＴＥを実行すると次の命令が実行される。

これは，この例外の起きた状態はＯＳサポート作業を行ない次の実行に進むためのものである。通常のソフトウェア割込み等がこれに当たる。これはフォーマット２となっている。

命令途中での例外用スタック・フォーマット（**図10-9-4**）は，10ワードの積上げを行ない，その命令自体のアドレスと内部レジスタ（ＭＰＵの）を４ワード含む。これはフォーマット９となっている。このスタック・フォーマットは，コ・プロセッサの命令の中断用である。

この他に，通常のＭＰＵの命令の途中で例外処理が発生し，それが即時に処理される場合には16ワード（フォーマット＄Ａ）か46ワード（フォーマット＄Ｂ）が作られる。

この区別は，その例外処理の発生が命令の境界上にあるか否かでＭＰＵが自動的に判断する。このスタックの内容については，ユーザーズ・マニュアルの第６章6.5を参照されたい。これらのスタック・フォーマットはＲＴＥ命令によって自動的に判断されて適切なリストアが行なわれる。

10-10　マルチＣＰＵのサポート

68000ファミリで，マルチＣＰＵのサポートは，一つのシステム上の共用のバス（グローバル・バス）上の資源（メモリ，ディスクＩ／Ｏなど）を各ＣＰＵは共用して働く。更に共用バスの部分以外に自分専用の（ローカル）メモリや入出力を持っているものと考えられる。

もし完全に一つのバス上に二つのプロセッサを置くと，各プロセッサは自分のプログラムをそのバスから読むだけで，バスの使用権を獲得しなければならないが，もう一つのプロセッサも同じことをしているため，待合いが生じ各プロセッサの機能は大幅に下がる。

そこで，各プロセッサに一定のTASKを処理できるだけのプログラム用及びデータ用のメモリを与え

図10-9-1　フォーマット$0-4ワード・スタック・フレーム

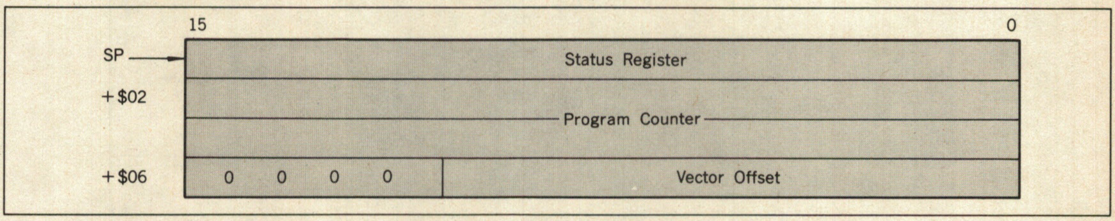


図10-9-2　フォーマット$1-スローアウェイ4ワード・スタック・フレーム

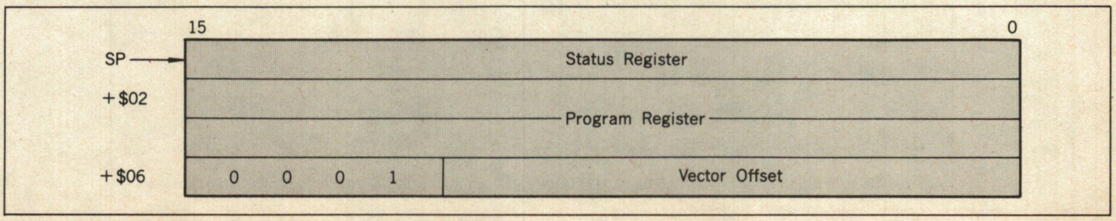


図10-9-3　フォーマット＄-6ワード・スタック・フレーム

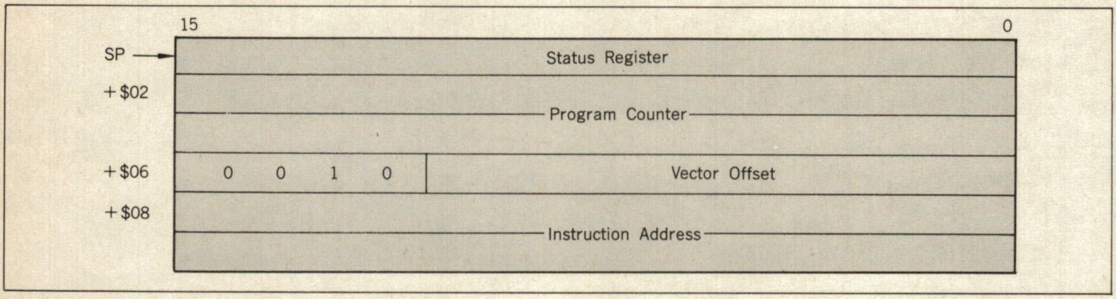


図10-9-4　フォーマット＄9-コ・プロセッサ「命令途中での例外」用スタック・フレーム(10ワード)

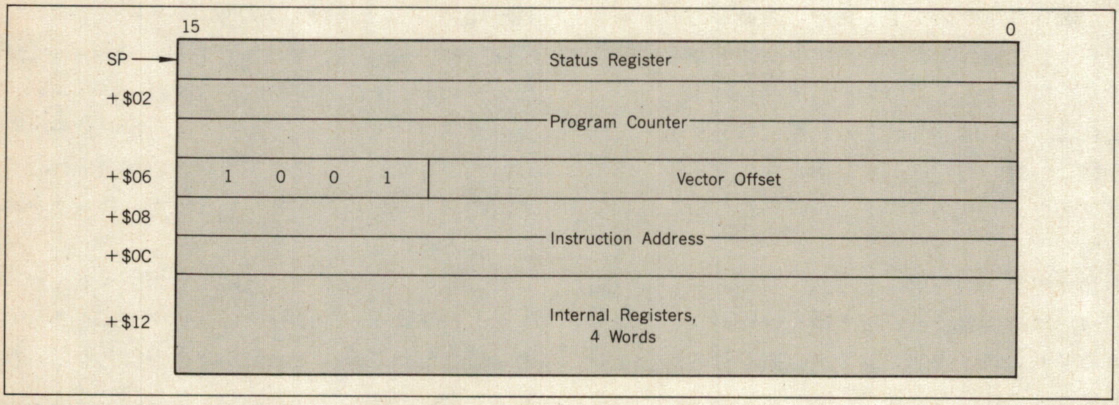


図10-9-5　フォーマット＄A-ショート・バス・サイクル・フォルト用スタック・フレーム(16ワード)

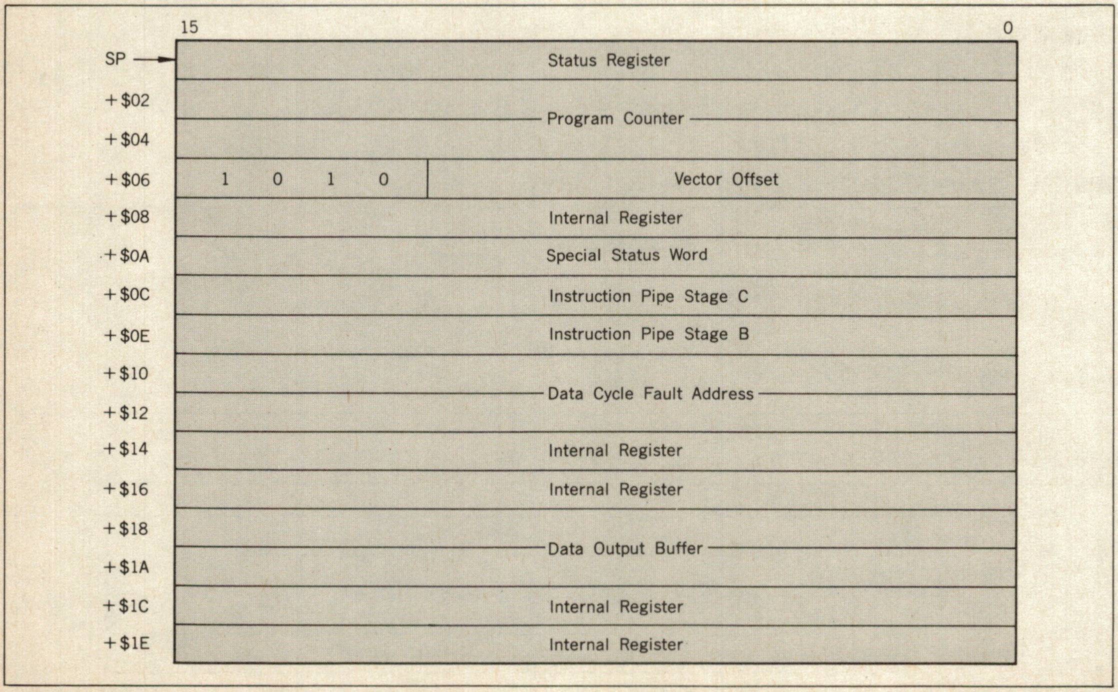
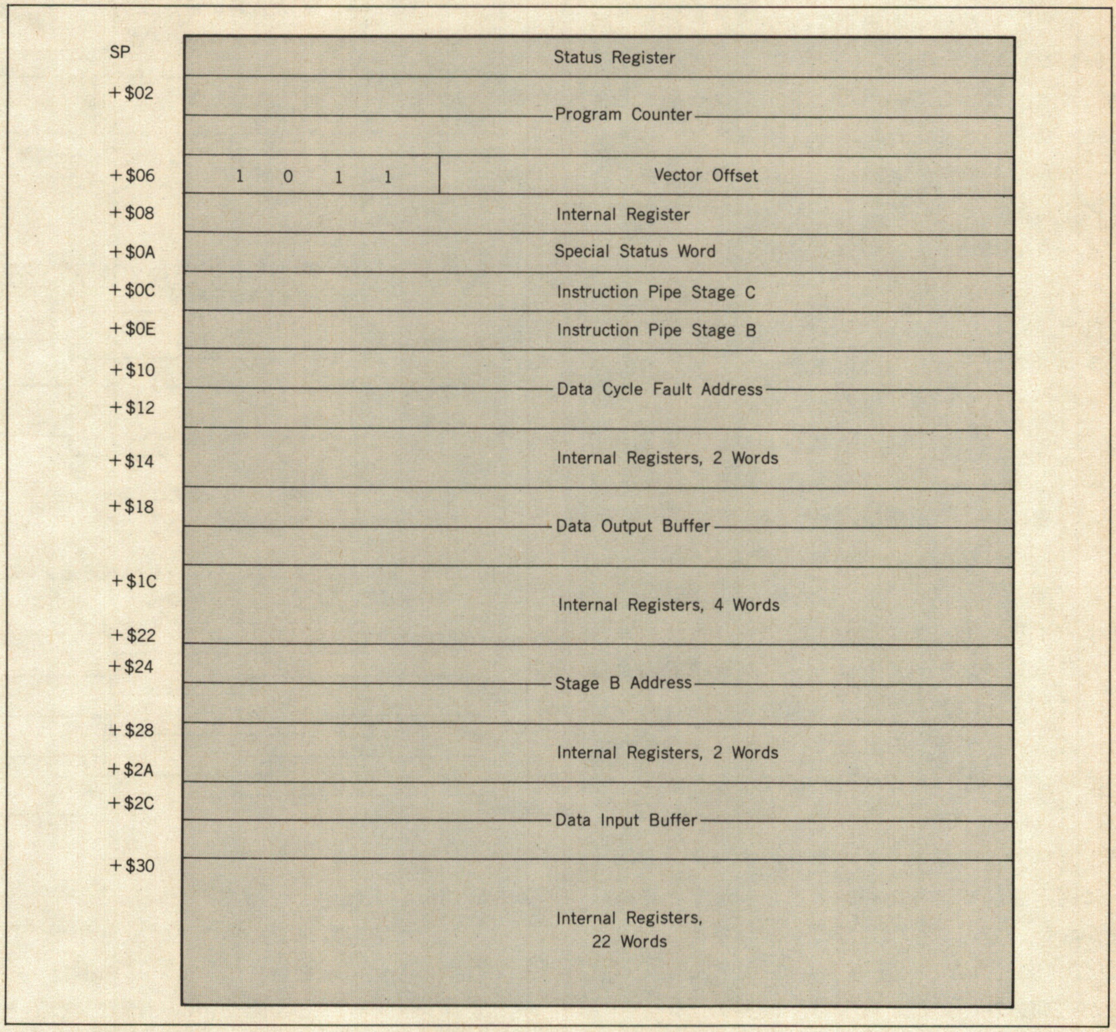


図10-9-6　フォーマット$B-ロング・バス・サイクル・フォルト用スタック・フレーム（46ワード）



そこで主たる処理を行ない，他のプロセッサとの通信，データの転送，プログラムの受領，指示，命令の受領等に共通領域のメモリやＩ／Ｏが使われる様にすれば，各プロセッサの共通バスへのアクセス頻度は下がる。共通の資源は本来の共通の目的に使われ，個々の資源は個々のプロセッサの実行に使われるという事になる。

68000ファミリでは，この様なマルチ・プロセッサの働きをサポートするために，幾つかの特殊機能を持つ命令を用意している。

この命令は68000ファミリの「ＴＡＳ」命令，68020／30の「ＣＡＳ，ＣＡＳ２」命令で，両方共リード・モディファイ・ライト命令である。

このリード・モディファイ・ライト命令というのは，あるオペランドのメモリの内容を読み，その後に書込みを行なうという処理を一つのメモリ・サイクルで実行する事を言う。この様な命令も，通常のシステムであれば読出したデータが書込みまでの間に変更される事は考えない。

ところが，マルチ・プロセッサの場合には，プロセッサ同志の競合が起きてデータが変更される事が充分考えられる。そのメモリのバイトが，フラグに

なっており，あるメモリ・ブロックの使用権をどちらのプロセッサが持つかを示す様な所であると，図**10-10-1**の様に，まず０でメモリ・ブロックがフリーであることを表わしていたとして，プロセッサＡとプロセッサＢが順にこれを読み，両方共，メモリ・ブロックが空きであると認識し，自分が使用することを表すために自己の認識コード（ＩＤ）を書き，そのメモリ・ブロックを両方が使用し始めるというシステム上のバグが起きる。

これを避けるために，このリード・モディファイ・ライト命令では，バスの使用権をロックし使用権の変更が行なわれない様にする必要がある。

このバスの使用権は，ＭＰＵ自体の「BR,BG,BGACK」の信号と，外部ロジックの働きで行なうので，これらにバスの使用権の移動（バス・アービトレイション）を行なわない様にしてやる必要がある。

バスが単純な68000／08／10／12では，このリード・モディファイ・ライト・サイクル（RMC）の間，ＡＳ（アドレス・ストローブ）が連続で出力される。このようにするとバスの使用権の実質的な変更は，ＡＳのネゲイトをもって行なうという規則に従って単純に行なわれる(図**10-10-2**)。

これは，68000／08／10／12で使えるこの種の命令は「ＴＡＳ」しかなく，この命令のオペランドはバイトに固定されているため，アドレスが変更される心配がないので容易な形で行なう事ができる。

68020／30ではこの「ＴＡＳ」に「CAS, CAS2」命令が追加された。この命令は，最大４ロング・ワードの転送を行なう。

更に，ダイナミック・バス・サイジングのため，ワード転送であっても，メモリの２ワードにまたがって行なわれる事があり，アドレスが一定であるという意味のＡＳによってロックをかける訳にはいかない。

68020では，ＡＳは各サイクルごとにどんな場合でも出力されてくる。その代り「RMC」という出力がMＰＵに設けられ，バスのアービトレイションをロックしなければならない様な場合にはこの信号を出力する。バスのアービトレイション回路は，使用権変更要求を受けても，ＲＭＣによりロックされている区間は要求に応じてはならない。

図10-10-1　あるメモリ中のフラッグ・バイトを二つのＭＰＵ（Ａ，Ｂ）が同時に変更した場合

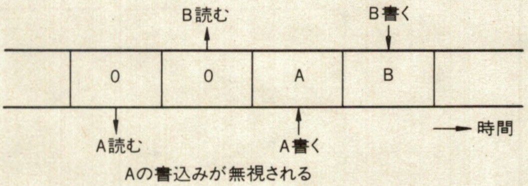


図10-10-2　68000／08／10／12の場合のRMCサイクルのタイミング（TAS命令時）

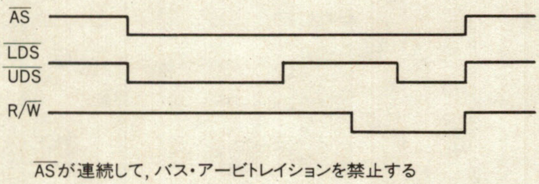
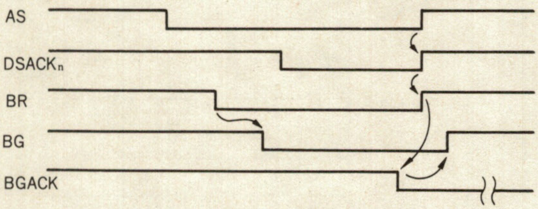


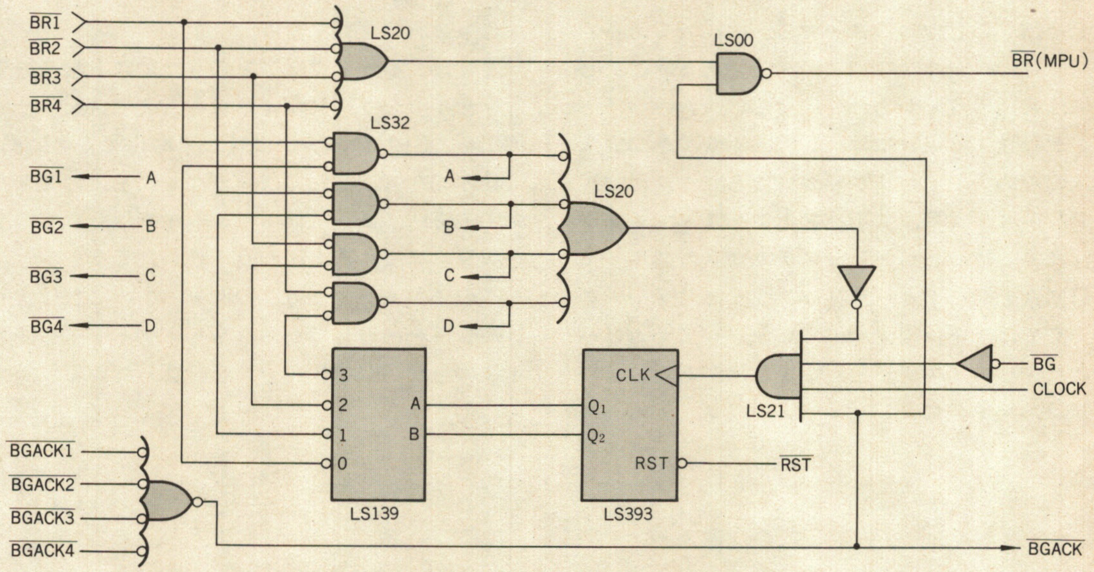
図10-10-3　68000ファミリのバス・アービトレーション・タイミング



バスの使用権を要求した他のプロセッサや，ＤＭＡ等のバスの使用権を持てるディバイス側は，バスが，４ロング・ワードの間使用権が得られなくともバス・エラーや，バッファ・オーバーフローを起こさない様に考慮しなければならない。特に，バス・マスタとなるディバイスが二つの時はまだ良いが（ＭＰＵとＤＭＡの時の場合)，三つ以上になると（ＭＰＵ×２＋ＤＭＡ＋α）単純な１ラインのアービトレイションや，プライオリティ・エンコーダによるアービトレイションは，最下位の要求者がバスをとれない時間がかなり長くなる。場合によっては数100msec以上という事もあり得るので，バスのアービタの設計には注意が必要である。

連続的に共用のメモリやその他の資源を使うシステムでは，優先順位のレベルをダイナミックに変更

図10-10-4　スキャン型ラウンド・ロビン・アービタ



このアービタは，バスの要求をカウンタによりスキャンして，始めに見つけたものにバスの使用権を与えるものである。バスの要求のない時は，スキャンは停止して，最後に受付けた次のレベルを指している。バスの要求が入ると，アービタはこのレベルより順次捜して行くのでバス要求自体は同一レベルになり，各要求元は平均してバスを使う権利を獲得できる

して，バスの使用度を平均化するラウンド・ロビン型のアービタが望ましいと思われる(**図10-10-4**)。

10-11　バス・アービトレイション

68000ファミリのバス・アービトレイションは基本的には同一で，ＭＰＵとしては自分と他の１レベルのアービトレイションしか行なわない。バス・マスタが多数存在する場合のマルチ・レベルのアービトレイションは外付の回路で行なう事になっている。

68020／30は，この外付アービタに対し，RMC信号によってアービトレイションの禁止が要求されるが，これ以外の信号の処理に関してはまったく同一と考えられる。

ＭＰＵの行なう手順としては，ＢＲ：バス・リクエストが，バスの要求者からＭＰＵに与えられる。そうするとＭＰＵはＢＧ：バス・ギャラントを出力し，次のＡＳのネゲイトで，バスの制御を外部に移す事を示す。外部の要求者は，このＢＧが返ってくるまで何もしてはいけない。ＢＧがきて，ＡＳがネゲイトされると次のバス・サイクルは行なわれず，アドレス関係，データ・バスは，ハイ・インピーダンスになっている。そこで，外部の要求者はBGACK(バス・ギャラント・アクノレッジ)をＭＰＵに与え，ＢＲをネゲイトする。

そこで，バスを要求してきた側ではバスの使用を開始する。

外部バス・マスタのバス使用が終了すると，要求者はBGACKをネゲイトし，ＭＰＵはこれを受けるとＭＰＵ自身がバス・マスタとなって，動作を継続する。

ＭＰＵが外部バスをアクセスできなくなると，たとえ68020／30の様にキャッシュを持っていても，オペランド・アクセスが待たされるため，実質的にＭＰＵは停止する。更にコ・プロセッサとの通信もバスを通じて行なっているので，この動作も停止する。

バス・アービタの設計には充分注意することが必要である。

以上はMPUと他のバス・マスタの場合であるので，MPUのローカルなバスの場合である。これとは別に，共用の資源があって，そこに複数のバス・マスタが存在する場合には，この共用資源を構成するバスのためのアービタが必要になる。これはマルチCPUシステム等，共用バス上にバス・マスタが多数存在する時，各CPUシステムから，共用メモリにDMAを起こすと，単純なプライオリティ・エンコーダでは，下位優先順位のCPUシステムはバスをとれずバス・エラーを起こす危険がある。このあたりは，マイクロコンピュータ用のシステム・バスでの最大の問題となる所で，この種のシステムでは，ラウンド・ロビン型のバス・アービタを採用される事をおすすめする。

これは，ハードウェア上のリクエスト線の優先順位をダイナミックに変化させる方法で共通メモリの使用を各CPUシステムに対し平均化する様に働く。

具体的な方法としては，カウンタを設けてスキャンする方法，最後に使ったレベルを保持して，これを減算して決める方法が良く使われる。

このスキャンする方法の例を示す。これは，MPUがBGを出力してからASをネゲイトする間のサイクルを利用してスキャンを行なっている。

第11章

コ・プロセッサ

11-1 コ・プロセッサに対する知識

コ・プロセッサは，MPUのある特定の機能を強化する目的で追加されるシステムの総称で，68020／30のコ・プロセッサは68020／30との間で自動的に対話動作を行ない68020／30と命令を共同して実行する。この動作は68020／30と基本的には非並行に実行される。

コ・プロセッサの機能は通常のI／Oディバイスとしても実現できるが，I／Oとした場合はある程度のソフトウェアのサポートが必要である。

コ・プロセッサをコ・プロセッサとして使うメリットは，I／Oと異なりプログラマから見ると，MPU自体がその機能を持っているようにみえる事である。したがって，プログラム自体にコ・プロセッサ命令を混在できる。

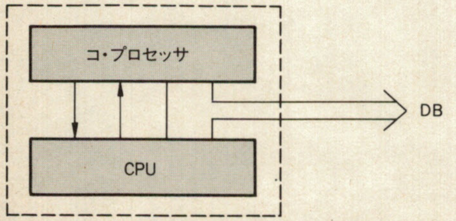
68020／30に接続されるコ・プロセッサに対し，ユーザーにより必要とする知識のレベルに差がある。

1)．モトローラ社，あるいは他のメーカーから販売されるコ・プロセッサを使うユーザーは，コ・プロセッサの機能と命令記述法を知っていれば充分でMPUとコ・プロセッサの通信の知識は不要である。

2)．システム・プログラマやエンジニアは，コ・プロセッサの概略の動作，例外発生の条件を知っている必要がある。主に，この章は，このレベルをサポートするようにしてある。

3)．コ・プロセッサの設計，あるいはコ・プロセッサ自体のデバッグを行なう人は，このコ・プロセッサ・インターフェースの詳細を知る必要がある。これはユーザーズ・マニュアルに記載されている。通信プロトコル，プリミティブの詳細をご覧いただきたい。

図11-1-1　MPUのイメージ



コ・プロセッサはプログラマのイメージとしてはMPUと同一に扱われる

11-2 コ・プロセッサとの通信

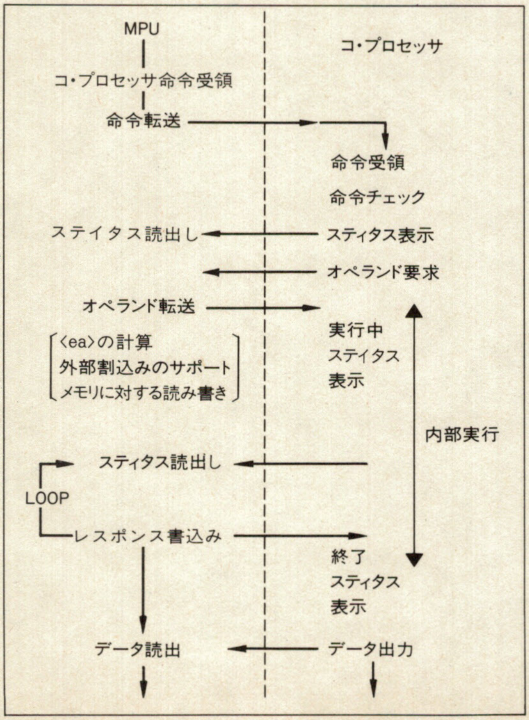
68020／30ではMPUの機能強化のために，ある特定の処理を受け持つコ・プロセッサが接続できるようになった。

このコ・プロセッサは，68020のユーザーズ・マニュアルに，「68020のコ・プロセッサ・インターフェースの規格を満足して接続されるもの」と定義されている。

これは，MPUとの間に予め定められた手順（プロトコル）によって通信を行ない，その通信によってコ・プロセッサが必要とするコマンドの受領，データの受渡し，場合により内部状態のセーブ，リストア（戻し）を実行する。

この通信は，プログラマが一切管理する必要がな

図11-2-1　コ・プロセッサとMPUの通信



く，コ・プロセッサが行なっている処理は，見かけ上，ＭＰＵが行なっているように見える事になる。

コ・プロセッサが通常のＩ／Ｏディバイスと異なるのはこの点であり，周辺Ｉ／Ｏならばそこにデータを与える，データを読出す，命令を与える等の処理をプログラマがソフトウェアにより管理しなければならないが，コ・プロセッサとして接続されているディバイスは，入出力に関しては，ＭＰＵがコ・プロセッサとの通信の中で自動的に行なうため，プログラムはディバイス管理の必要はない。

逆に，コ・プロセッサは，68020コ・プロセッサ・インターフェースの規格を満足する入出力形態を持ち，自分に対するコマンドやデータはこの通信を通じて受け，処理を行なった結果を送らなければならない。また，このコ・プロセッサ通信プロトコルによって与えられる命令が，ＭＰＵのコ・プロセッサ・サポートの機能として取入れられるような形式を保つ事が必要である。

したがって，68020／30にコ・プロセッサを接続しようという時，接続されるコ・プロセッサは回路上68020／30との専用通信回路をサポートする必要があり，ここがコ・プロセッサ本体の処理部と68020／30を接続する。

11-3　コ・プロセッサ・インターフェース

ここの部分は，まず68020／30との非同期のインターフェースである必要がある。この点は，通常のＩ／Ｏと変わりない。そして32ビットなり16ビットなりのデータ・バスによってＭＰＵと接続され，DSACKxによって，アクノレッジを返す点も同一である。そして内部に16ワード（16ビット）のレジスタを持ち(最低８個)，各レジスタ機能は予め決められているコ・プロセッサの命令の受領，補助処理の要求状況の表示あるいはデータの受渡しを行なうものとされている。このレジスタ群はコ・プロセッサ・インターフェース・レジスタ（ＣＩＲ）と呼ばれ，コ・プロセッサの動作はここから規定され，このＣＩＲを持つものがコ・プロセッサだと言える。

このＣＩＲは，予め68020／30のアドレス空間内に特定の位置が決められている。その位置は，コ・プロセッサとの通信，コ・プロセッサの実行はＭＰＵの処理自体と同一に考えるので，当然，ＣＰＵ空間内である（ＦＣコード＝７，全部Ｈ)。

図11-3-1　コ・プロセッサ・インターフェース・レジスタ(CIR)

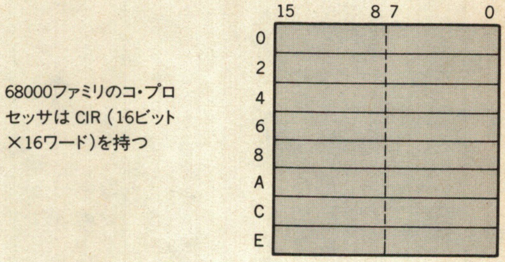
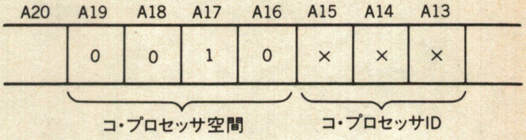


図11-3-2　コ・プロセッサ空間とコ・プロセッサID



そして，ＣＰＵ空間のタイプ・フィールド（Ａ19～Ａ16）によって，コ・プロセッサ空間と決められている部分に接続しなければいけない（図11-3-2)。このアドレス以外に接続した場合は，コ・プロセッサ・インターフェースとの通信が行なえない。このアドレス・エリア中，更にＡ15～Ａ13の３ビットがコ・プロセッサＩＤとして８個のプロセッサの指定コードとして使われる。

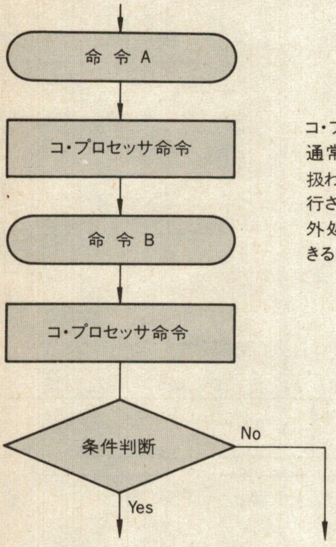
また，ページ・モードMMU (68851)は，必ずこのＩＤ番号０への接続となる。

これは，このディバイスがＦＣコードやＣＰＵ空間，タイプ・フィールド，コ・プロセッサＩＤを内部でデコードし，更にこれらの信号を内部で変換して，各ページ空間に拡張する際に使っているからである。

Ａ４～Ａ０が正しくコ・プロセッサ内のＣＩＲのデコーダに接続されてコ・プロセッサは68020／30と通信することができる。

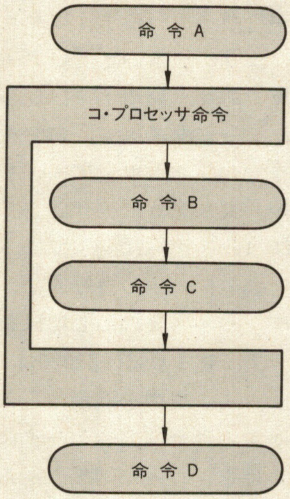
こうして，ハード上，正しくコ・プロセッサが接続されると，プログラマはコ・プロセッサが定義している拡張命令をソフトウェアの中に直ちに記入す

図11-3-3　コ・プロセッサの命令フロー



コ・プロセッサ命令も，通常の命令と同様に扱われ図の様に遂時実行されていく。ただし例外処理は別途実行できる

図11-3-4　並列処理の場合



コ・プロセッサ命令は条件によってはプログラム・フローと並行に処理を行なわせる事が出来る

れば実行される。この時，コ・プロセッサの命令も通常の68020の命令と命令の流れ，命令ストリームを構成する。MPUは見かけ上，順に1命令ずつ実行していくというノイマン型CPUの動きを，コ・プロセッサの命令を含めて行なう。つまり，コ・プロセッサ命令も，単なる1命令として処理される事になり，前の命令が終了しなければコ・プロセッサ命令は実行されず，コ・プロセッサの命令，コ・プロセッサの処理が終わらなければ次の命令が実行される事もない。もちろん，使い方及び設計によってはコ・プロセッサが実行中であっても，MPUを解放して次の命令に進ませる事もできるが，この場合，データ処理上に混乱が起きないようにするのは，コ・プロセッサの設計者及びその動作を行なわせるソフトウェアの作成者の責任であるとされている。

通常は，コ・プロセッサの命令はMPUの命令と同様，プリ・フェッチ，コード自体の予備チェック等が行なわれたとしても1ステップずつ進む。したがって，SR内のトレース・フラグを制御しトレース実行をさせた場合，ステップで実行する事も可能である。つまり，MPUから見てコ・プロセッサ命令は他の一般の命令と同様，ステップ実行が可能である。これは，1ステップ・トレース，命令フローの変更点までのトレースいずれでも行なえ，またコ・プロセッサ命令の実行中に例外処理が発生した場合には，トレースの処理が打切られる点も通常の命令と同一である。この形式はコ・プロセッサの内部処理がほとんど終わり，MPUのサポートは不要とMPUを解放した場合でも，トレース・モードにいる時は完全に命令終了まで待ってトレースを行なうので，更に完全なものとなっている。

11-4　コ・プロセッサと外部割込み

コ・プロセッサの動作の中で，一般の命令と若干異なった動作をする点がある。それは割込みに対しての処理で，ソフトウェアによる割込みはコ・プロセッサの命令を実行中という事からコ・プロセッサ自体が発する例外処理要求しかない。

しかし，外部からの割込みは内部の動作とは一般的に非同期に発生する。そこで，もしコ・プロセッサの処理が一般の68020／30の命令より長くかかる場合には，割込みを処理しきれなくなる場合が考えられる。また，このようなケースがむしろ一般的であると考えられる。そこで，コ・プロセッサとCIRで通信中にコ・プロセッサは「今MPUの補助を必要としていない」というコードを出力し，MPU

はこのステータスを受付けると，受けとったという応答を返す前に，もし受付けられている割込みがあればこれを処理する。

この時には，命令実行途中での例外という事で，10ワードをスタック上にセーブし，割込みの処理が終了して「RTE」命令で元に戻ると，先のペンディングになっている応答を行ない，またコ・プロセッサのステータスを読みに行く。MPUがCIRを通じてコ・プロセッサと通信を行なっている時，CPU空間は高い頻度でコ・プロセッサと通信を行なうために使われ，その間をぬって割込み処理を実行している事になる。

11-5　コ・プロセッサのセーブ／リストア

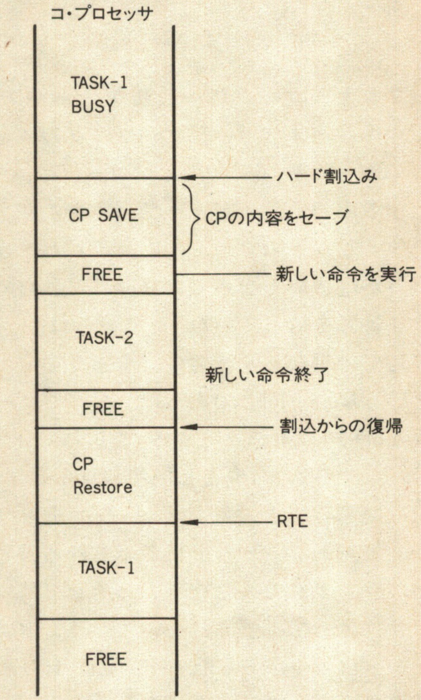
割込みによって起動された新しいタスクが，現在処理を行なっているタスクよりも優先順位の高いタスクで，即，実行する必要がある場合，そのタスク内の命令ストリーム中にコ・プロセッサ命令が出てきて，あるコ・プロセッサ命令が実行中なのに，同じコ・プロセッサに別のあるいは似た処理を現在の処理よりも高い優先順位で要求する事もあり得る。

この場合，コ・プロセッサは，ある命令を実行中で正しく処理終了の手続が行なわれていないのにかかわらず，新たなコマンドがコマンド・レジスタに書込まれる事になる。

これに対し，コ・プロセッサはMPUへの反応として命令受領，他の正常な応答でなく，「命令実行途中での例外発生」を応答し，MPUに後からのコ・プロセッサ命令を実行前に中断させ(アボートさせ) OSのエラー処理に制御を移す。OSは，もしコ・プロセッサが「cpSAVE, cpRESTORE」命令，つまり，コ・プロセッサ自体の内部処理を中断しての内部状態の保存，及び保存された内部状態の再設定及び継続実行の機能を持っていれば（持っていないとOSの処理は非常に複雑化する)，コ・プロセッサに実行を中断して内部状態をセーブするという命令を発し，内部のデータをデータ・スタック上なり特定のエリアなりへ退避させ，コ・プロセッサをフリーの状態にする。それから先に例外処理になってしまったコ・プロセッサ命令を実行するため，「RTE」を実行し中断した命令を再実行する。このようにすれば，優先順位の高いタスクのコ・プロセッサの使用が可能になる。ただし，この処理もネスティング（多重化）する恐れがあるため，OSのコ・プロセッサ・サポート部分には充分注意する必要がある。

この優先タスクが終了し，元のタスクへ戻る前には必ず，OSはcpSAVEを実行したか否かをチェックしなければならない。もしも，cpSAVEが行なわれている時は，コ・プロセッサの状態をチェックしフリーになっていれば，cpRESTOREにより元に戻してから，外部割込みを元に戻すためのRTE命令を実行し，先にペンディングとしたCIRへの応答を行ない次の内部チェックに移る(図11-5-1)。

図11-5-1　コ・プロセッサのプログラムフロー



コ・プロセッサに対して，cp SAVE, cp Restoreを用いて，タスクの切り替えを行ない，優先タスクのコ・プロセッサ命令を先に実行した場合のコ・プロセッサの状況

11-6　コ・プロセッサ命令

コ・プロセッサ命令は，最初から68000ファミリに，予め作り込まれていたオペコードの空きエリアを使って作られる。68000のオペコードは，オペワード16ビット中の上位4ビットで（命令の章を参照）この上位4ビットのコードの内，＄Fと＄Aは特に別の例外処理扱いとなっており，各々独自の例外処理ベクタ番号を持っていた。その意味は，68000／08／10／12では単なる不法命令，つまり命令コードがないという場合と同じであった。

それが68020／30では，この内，＄Fの命令コードがコ・プロセッサ系の命令として使われる事になった。これは，MPUにこの命令はコ・プロセッサ用の命令だからビット9～ビット11で示される番号のコ・プロセッサをアクセスしなさい，そして特にコ・プロセッサによって拡張されないオペランドのアドレス・モードはビット5～ビット0の6ビットに入っているため，これを使う準備をしなさい，という意味を持つ。またビット8～ビット6には，コ・プロセッサとMPUの双方に，これから命令する命令の系統を指示するコードが入っていて，単なる演算か条件判断か，実行の中断，退避か，元に戻す事を要求しているのかを知らせる。

MPUは，この命令ワードを見つけると次の命令ワードがコ・プロセッサに対する命令ワードという事で，これをコ・プロセッサのCIRの内，コマンド用のレジスタに書込む。

もし，このメモリ・サイクルで，バス・エラーが発生するとMPUは該当するコ・プロセッサがないと判断し，通常の空きオペコード＄Fに対する例外処理を発生する。

コ・プロセッサは，もし自分自身が他の命令の実行中なら，前述の割込みの項で説明したように，命令途中での例外処理を要求し，今回の要求は一時，待たされる。

そうでなければ，このMPUから与えられたコマンドをデコードして実行にかかる。

このコ・プロセッサ命令は，ディバイス・メーカーがコ・プロセッサを作った時に，その命令を含むアセンブラを作ってくれれば問題はない。ユーザーが作ったコ・プロセッサ，あるいはコ・プロセッサ命令が入っていないアセンブラの場合は，ユーザーがマクロ定義等で作らなければならない。

この時には，このMPUに対するオペコード中〈ea〉の入る6ビット〔0：5〕以外は，特定のプロセッサに対しては固定であり，6ビットの部分はモード3ビットとレジスタ3ビットに分かれる基本形であれば容易にコーディングできる。

第2のコ・プロセッサに対する命令ワードはコ・プロセッサの一つ一つの動作に対する命令で，コ・プロセッサにより定義されているため各動作に対しては固定と考えられる。その後ろには必要なだけのオペランド・データ，コ・プロセッサが必要とする拡張ワードとなっている。この部分は，コ・プロセッサがMPUとの通信によって要求してくるデータを順に並べるため，コ・プロセッサによって，予め決めてある順序に従う（図11-6-1）。

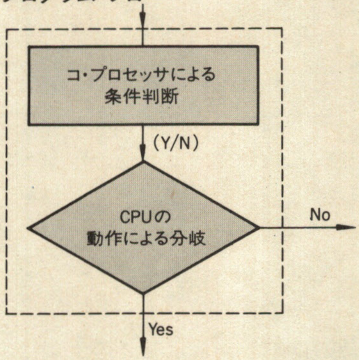
11-7　コ・プロセッサの命令の実行

このコマンド・ワードにより，MPUとコ・プロセッサの分担が決まる。これは，コ・プロセッサ命令を実行してゆく時にデータの読出し，または書込みのアドレス指定は，通常のアドレス・モードを使

図11-6-1　コ・プロセッサの汎用命令のフォーマット（cpGEN）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			0	0	0	Effective Address					
Coprocessor Command															
Optional Effective Address or Coprocessor Defined Extension Words															

図11-7-1　コ・プロセッサに条件判断を求めた場合の
プログラム・フロー



って行なうために，当然ながら〈ea〉の計算とオペランドのメモリ・サイクル，つまり実際のリード／ライト作業はMPUが行ない，MPUを通じて与えられたデータの処理，演算，実行，結果の算出はコ・プロセッサが行なう。そして，コ・プロセッサが出した結果によるデータの転送はMPUが行なう。

この他，各種の異常事態に対しての例外処理への移行はコ・プロセッサの要請によりMPUが行なう。

こうした役割の分担によりコ・プロセッサへの命令は四つに分けられる。

その1は通常のコ・プロセッサに対する演算，処理の要求で，コ・プロセッサ本来の機能を直接的に利用する命令部で，マニュアルにcpGENとして説明されている。ここでいう「GEN」とはGENERAL＝一般，普通の略号で一般の命令を意味する。

その2は，コ・プロセッサに，分岐，トラップ等を起こすか，起こさないかの判断を要求する命令部で，コ・プロセッサはMPUから判断条件を受け演算，判定を行ない，その判定結果をMPUに返し，MPUは各種の「Bcc, DBcc, TRAPcc, Scc」の条件判断の判断部をコ・プロセッサに任せた形となりMPUは通常の上記の命令と同様の動作を行なう。

その3と4は，先に説明した「cpSAVE，cpRESTORE」で，コ・プロセッサの命令実行の中断，内容の退避と，退避した内容を元に戻しての中途からの再実行となる。

コ・プロセッサとMPUは，これら命令の種類をコ・プロセッサ命令オペコードの最初のワードのビット8～6で受領する。

そして，MPUはこのコードによって，コ・プロセッサ命令の第2ワードを自動的にコマンド用CIRに書くか，コンディションCIRに書くかを決めてくれる。

MPUから，コ・プロセッサがサービスを受けるのは同じで，〈ea〉を計算しデータを転送するのはMPUが行なう。多少違うのは，一般の命令では単に命令終了だけだが，条件判断の場合は終了と同時に判断の結果がTRUEかFALSEかを返してやる点である。このあたりの通信の詳細は，通常のユーザーや，プログラマには見えないから，余り細かい点は知らなくても済む。ただしコ・プロセッサとの動作のデバッグ時や，コ・プロセッサを新たに設計する場合には，この辺の事は熟知している必要がありソフトウェアの点から見れば新たなコ・プロセッサ命令をマクロにて作成する場合に，コ・プロセッサの仕様と共に充分理解する必要がある。

その場合は，各通信の状態を説明している「コ・プロセッサの命令の種類」(ユーザーズ・マニュアル8.3) を参照されたい。

11-8　コ・プロセッサの実行の中断，再開

先に割込みとの関係で，コ・プロセッサの処理が中断再開される点を説明したが，プログラマが注意するべきは，転送が全てロング・ワード形式で行なわれ，最初の1ロング・ワードの上位1ワードはフォーマット・コード1バイトと，何バイトのデータの転送を行なう必要があるかという転送長1バイトで構成されている事である。cpSAVE実行時には，このフォーマット自体は評価されず，転送長のみが使われるが，転送長が4の倍数つまり転送長データのビット0と1が，Lでないとフォーマット・エラーという例外処理を引き起こす。

cpRESTOREを行ない，中断された状態を元に戻そうという時は，MPUは，まず，このフォーマットと，転送長の入っているワードを読出して，コ・プロセッサに転送する。コ・プロセッサは，このフォーマット・コードを評価して，エラーがあれば例

表11-8-1 コ・プロセッサ・フォーマット・ワードのデコーディング

フォーマット・コード	長さ	意味
00	xx	Empty/Reset
01	xx	Not Ready, Come Again
02	xx	Invalid Format
03-OF	xx	Undefined, Reserved
10-FF	Length	Valid Format, Coprocessor Defined

外処理の要求を，なければ内部データの転送受入れの準備をする。

このフォーマット・コードは，このリストア命令受領後，MPUに返すが，この時いくつかのコードが決められており，MPUはこのコードにより，転送を延期したり，例外処理を起こしたりする（**表11-8-1**）。

また，プログラマが知っておくべき事は，先の割込みのところで説明したように，このセーブ／リストアの動作はOSが行なうものでOSの特権命令となっており，ユーザー・モードで実行しようとすると，プリビレッジ・バイオレーション例外処理を引き起こす事である。

11-9　コ・プロセッサ発生の例外処理

コ・プロセッサは，MPUに対し例外処理を要求できる。これは通常のI／Oディバイスが割込みをかけるのと同じ動作をするが，割込みのように非同期には行なえず，MPUがコ・プロセッサのステータスを読みにきた時にしか起こらない。この時，コ・プロセッサはMPUの内部発生の例外処理と同様に命令実行前の例外処理（命令が実行不可能であるかコ・プロセッサへの不当命令か，といった場合）を起こせるし，命令実行途中の例外処理（バス・エラー等と同様），命令実行後の例外処理(TRAP, CHK等と同様)も起こせる。

この時，コ・プロセッサは各例外処理の指定と共に処理先のベクタを指定でき，ベクタ番号を指定して，それぞれの場合に適した例外処理を要求できる。

また，これらの処理により，何らかのMPUのサポートを受けた後，RTE命令によって元の実行状態に戻れる。

11-10　コ・プロセッサとの通信のデバッグ

コ・プロセッサは，CPU空間に配置されるのでMPUはソフト的にコ・プロセッサをアクセスする場合は「MOVES」命令により他の空間へのアクセスを実行すればよい。そして，コ・プロセッサのCIRを決められた手順に従い，アクセスしていく事によって，コ・プロセッサ・インターフェースの動作をシミュレートできる。

この通信の詳細，レスポンスの手順は，ユーザーズ・マニュアルに譲るが，コマンド・コンディション・コントロール等，コ・プロセッサの動作を要求する通信を送った場合，必ず応答レジスタを読んでプロトコル上のハンドシェークを行なう事になっている点に注意されたい。例外として，コ・プロセッサはデータの転送をブロックで行なう時のみ応答を必要としないが，一般の通信では必ず必要とされている。

ただし，このモードはシステムの他のプログラムが，コ・プロセッサに対し処理を要求していない事を確認の上で行なう必要がある。そうでないとコ・プロセッサの動作が異常になる事は充分あり得る。

このデバッグの手順は，各々コ・プロセッサをI／Oディバイスとして接続した時と同一である。68020／30のコ・プロセッサは回路上は一般のI／Oと何ら変わらず，定められた通信プロトコルに従うだけで，MPUとのソフトによらない通信をサポートしているだけである。もちろん，ソフトウェアにより動作をシミュレートして単なるI／Oとしても使える。この時は，MPUが通信プリミティブの内容として自動的に解析，実行している部分をソフトウェアによって実現すれば良い訳で，浮動小数点演算のコ・プロセッサ68881の場合のこの様なソフトウェアの例は，オリジナルのモトローラ社より提供され下位の68000ファミリへの接続をサポートしている。

更に注意すべき点は，このコ・プロセッサのアドレスである。コ・プロセッサが接続されているシステムではFC0，1，2により，CPU空間と他のメモリ空間は分離されていなければいけないが，よく注

意しないと，コ・プロセッサへのアクセスが，同時にメモリへのアクセスにもなってしまう事がある。

11-11 MC68881/2

このコ・プロセッサは数値演算専用のコ・プロセッサで，68020に浮動小数点の単純計算，初等関数の計算能力を与えるものである。この素子は，68020のコ・プロセッサ・インターフェースを通じてMPUと交信し，その接続は容易であり，その例は第14章の実例にも示されている。このコ・プロセッサはその性質上，何らの外部制御，外部メモリ，等々の資源を必要としない。そこで，単にMPUと接続を行なうだけで，MPU内に拡張されたレジスタが生じた様に扱う事ができる。

このフローティング・ポイント・コ・プロセッサ（FPCP）の計算範囲は64ビットの実数部，正負を表す1ビット，指数部15ビットとなっている。また演算ユニットの演算長は67ビットあり，これにより計算時間を短縮している。演算データ用のレジスタ（80ビット）は全部で8本ある。

このレジスタは数値演算の命令では，MPUのデータ・レジスタと同様に結果の入るべき所，つまりディストネイションとして用いられる。演算のソース側は68020と接続されていれば，68020がソースとなるデータをコ・プロセッサ・インターフェースを通じてFPCPに与えるので，68020の全てのアドレス・モードを用いる事ができる。もちろん，このFPnレジスタも演算ソース側となれる。

このレジスタ長は80ビットあるが，常にこの長さの計算を行なうわけではない。計算長は，MPUのアセンブラ・ニーモニックで使用したのと同様の方法で演算長を指定できる。つまりニーモニックに続いて（.）ピリオドを記し，更に1文字で計算長を表す。この計算長は，MPUでは，B, W, Lの3種とアセンブラに表記はLであるが実際にはE（Extra Long）とでも示すべき例外1種であったが，FPCPでは図11-11-2に示す7種がある。B, W, Lは通常のMPUによる演算の場合と同様に，最上位ビットを正負の記号ビットとした整数値であり，これは

図11-11-1 拡張されるレジスタ

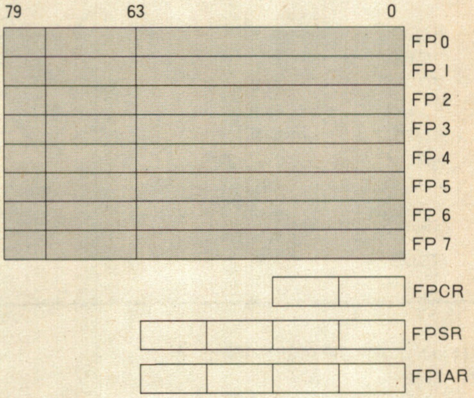


図11-11-2 演算長指定のポスト文字

B	バイト（8ビット整数，含±）
W	ワード（16ビット整数，含±）
L	ロング・ワード（32ビット整数，含±）
S	実数24（含±）指数部 8
D	〃 53（含±） 〃 11
X	〃 64（含±） 〃 15
P	10進実数部17桁 指数3桁

FPCP内に入るとFPCP内の標準の浮動小数点付の数値に変換されて用いられる。

通常の浮動小数点付の数値はS, D, Xの3種が用いられる。最後のPは（Packed－Decimal String Real）の事であり，これは実数部を10進数で17桁，指数部も10進数で3桁分の表記の事であり，FPCPとの入出力等に用いられる。これを用いるとFORTRANのFフォーマットでFPCPと入出力が行なえる。

例）FMOVE. P FP0, AA {#－8}

FP0の中のフローティング・バイナリの数値データを10進数に変換して下8桁分をラベルAAの示すメモリに入る。

11-11-1 関数

FPCPで計算できる通常の数値を扱う命令は図11-11-3に示す通りであり，この各々が先に示したデータ長をとる事ができる。

この内，特に単精度の乗除算は計算時間が短くなる様に作られている。この二つを除けば他の計算は

図11-11-3 FPCPの基本演算命令

FADD	加算
FCMP	比較
FDIV	除算
FMOD	剰余
FMUL	集算
FREM	IEEE, 剰余
FSCALE	指数計算
FSGLDIV	単精度除算
FSGLMUL	単精度乗算
FSUB	減算

図11-11-4 FPCPの初等関数命令

FABS	絶対値
FACOS	アークCOS
FASIN	アークSIN
FATAN	アークTAN
FATANH	ハイパブリック・アークTAN
FCOS	COS
FCOSH	ハイパブリックCOS
FCOSH	ハイパブリックCOS
FETOX	ex
FETOXM 1	$e^x - 1$
FGETEXP	指数
FGETMAN	実数
FINT	整数
FINTRZ	小数部切捨, 整数
FLOGN	ln(X)
FLOGNP 1	ln(X+1)
FLOG10	$\log_{10}X$
FLOG 2	$\log_2 X$
FNEG	Negate
FSIN	SIN
FSINH	ハイパブリックSIN
FTAN	TAN
FTANH	ハイパブリックTAN
FTENTOX	10^x
FTWOTOX	Z^x
FSINCOS	SIN&COS

全て80ビットのX精度で行なわれる。

このFPCPの主目的となる関数機能を**図11-11-4**に示す。

最後のFSINCOSは，ＳＩＮとＣＯＳの値を同時に得られる。これらの関数のほとんど全てが，ある特定の一つの値を加工するものであるが，この様な場合，ＭＰＵは，一つのオペランドしか要求しないのが一般であるが，FPCPに於ては，結果の入れられるレジスタはFPCP内のFPnでなければならないので，これらの関数命令は全て，オペランドを二つ(FSINCOSでは三つ)要求し，データを転送する途中で加工し，結果をFPnに入れる事ができる。

図11-11-5 FPCPの分岐命令

FBcc	条件分岐
FDBcc	データ・レジスタ＝－1　OR　条件分岐
FNOP	無効命令
FScc	フラグ・セット
FTST	FPSRのCCRセット
FTRAPcc	条件により例外処理へ移行

11-11-2　条件判断

プログラムの流れは，数値計算を行なった結果により変えられる事は多々あるが，FPCPはこれをサポートするために**図11-11-5**の様な条件判断命令，またはフラグ・セット命令を持っていて，ＭＰＵと協力して，これを実行する。この時FPCPはＭＰＵのコ・プロセッサ命令としての条件判断要求に応えてステータスを返す(詳細は11-7項参照)。これをＭＰＵは判断して，これらの命令を実行する。ここで条件区分の「cc」部はＭＰＵの条件判断の場合と同一の条件をあてはめる事ができる。

11-11-3　多重使用

プログラムの優先順位により，FPCPの動作を中断し，外部にセーブし，また後で，再設定を行なって実行を継続するためのセーブ，リストアの機能はFSAVE, FRESTOREとして持っている。

11-11-4　FPCPの制御

FPCRはFPCPのコントロール・レジスタで各種の条件により例外処理への移行を起こさせるか禁止しておくかのためのビット，８ビットと演算の有効長及び下位のまるめ方の指示に用いられるビット(各２ビットがある)がある。

FPSRはFPCPの実行状況またはエラーの状況を表示する。ここにはFPCPのCCR (N, Z, I, NANの４ビット，Ｎ：負，Ｚ：０，Ｉ：無限値，NAN：非数値)，商のバイト，例外処理への移行原因の表示

を行なうバイトがある。

11-11-5　68881と68882

この2つのFPCPはハード的，またソフト的にもまったく同一のものである。しかし，68881は，コ・プロセッサ・インターフェース部にデータを渡す部分が若干遅いという評価が出たために，これを改善して，トータルの性能を向上させたものが68882である。

このディバイスの詳細は，FPCPのユーザーズ・マニュアルを御参照（日本語訳が小社より刊行されている）いただきたい。なお，追加として，モトローラ社のデータ・シートに記載されている，この68881の計算時間の例を数例示す。表示の数字はクロック数であり，カッコ内が16.67MHzで実行した時の実行時間である（表11-11-1）。

11-11-6　68000/10での使用

FPCPを68000他のプロセッサで使用する事は不可能ではない。ただしFPCPはコ・プロセッサ・インターフェースを通じて68020の内部のレジスタや，コントロールと直に通信する様に作られているのでこれをシミュレートするソフトウェアが必要となる。

これは，先に説明したコ・プロセッサ・インターフェースをソフトウェアで実行すれば良い。FPCPの命令は全てライン＄Fエミュレータにより例外処理に移されるので，この内で，シミュレートすれば68020のソフトウェアとの互換性も保たれる。

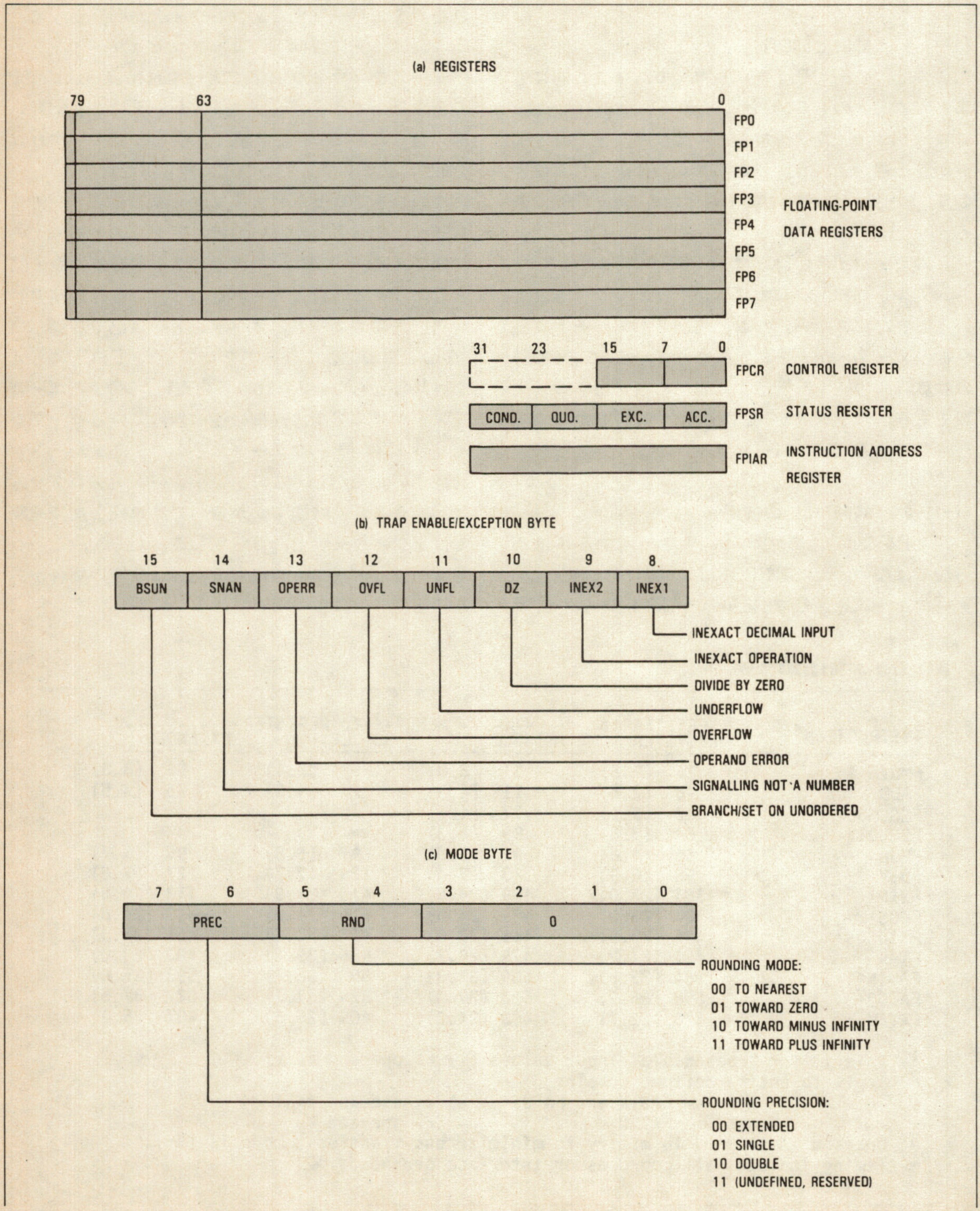
このコ・プロセッサ・インターフェース・シミュレータ上で最も難かしいのは，アドレス・モードの評価である。FPCPは，そのオペランド評価は全てMPUのサービスによっているので，これは，MPUが自己の命令コードのオペランドを自己で評価して<ea>を計算し，これをアクセスして，FPCPに与えなければならない。それ以外の点は，68020のユーザーズ・マニュアルに詳細に述べられているコ・プロセッサ・インターフェースの手順をシミュレートすればよい。ただし，このシミュレート・プログラムは，結構大きく時間もかかるので，単純な加減乗除ではソフトウェアで計算してしまった方が，このシミュレート・ソフトを用いる場合より速い場合があり得る。

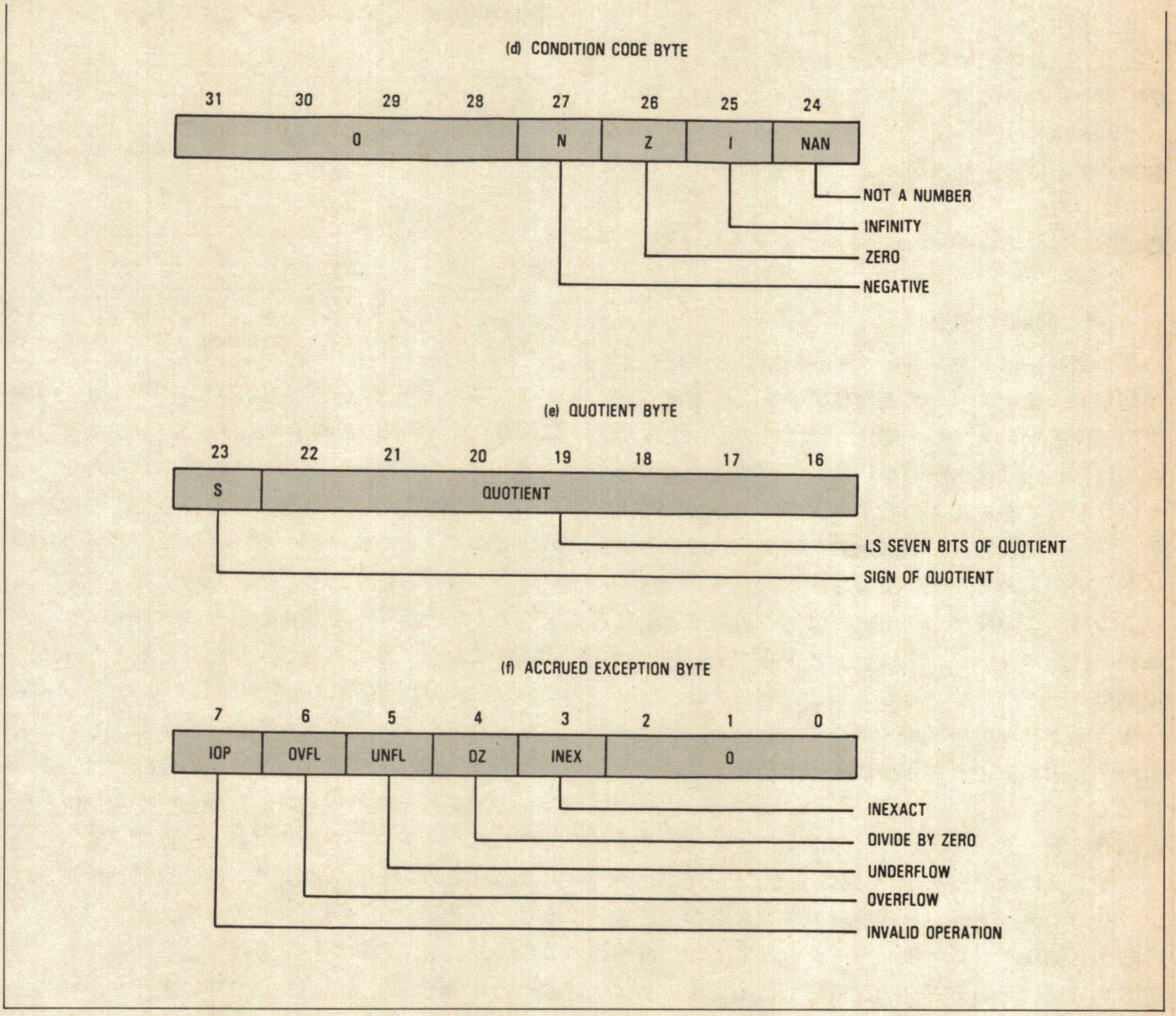
表11-11-1 68881計算時間例

INSTRUCTION[4]	REGISTER to REGISTER	MEMORY[3] to REGISTER SINGLE	DOUBLE	EXTENDED
FMOVE In	26 (1.5)	51 (3.0)	57 (3.4)	55 (3.3)
FADD/FSUB	46 (2.8)	71 (4.3)	77 (4.6)	75 (4.5)
FSGLMUL	52 (3.1)	77 (4.6)	--	--
FSGLDIV	64 (3.8)	89 (5.3)	--	--
FMUL	66 (4.0)	91 (5.5)	97 (5.8)	95 (5.7)
FDIV	98 (5.9)	123 (7.4)	129 (7.7)	127 (7.6)
FSIN[1]	384 (23.0)	409 (24.5)	415 (24.9)	413 (24.8)
FCOS[1]	384 (23.0)	409 (24.5)	415 (24.9)	413 (24.8)
FSINCOS[1]	426 (25.5)	451 (27.0)	457 (27.4)	455 (27.3)
FTAN[1]	454 (27.2)	479 (28.7)	485 (29.1)	483 (29.0)
FASIN[2]	564 (33.8)	589 (35.3)	595 (35.7)	593 (35.6)
FACOS[2]	598 (35.9)	623 (37.4)	629 (37.7)	627 (37.6)
FATAN[2]	378 (22.7)	403 (24.2)	409 (25.5)	407 (25.4)

1 This is for "reasonable" input values (-π <= operand <= π).
2 Apply to entire defined domain:
FASIN and FACOS　-1 <= operand <= 1
FATAN　- <= operand <=
3 Operand is aligned in memory to minimize bus transfers, mode is (An).
4 Timing includes all coprocessor interface overhead.

図11-11-6 68881プログラミング・モデル





11-12　MC68851

PMMUは，コ・プロセッサでもある。外付けのディバイスとしてのPMMUは，MPUの補助作業を円滑に進める為に，そのMPUとの通信はすべてコ・プロセッサ・インターフェースを通じ行なっている。

これは，通常のメモリ・アクセスはPMMUの変換の対象となるが，CPU空間へのアクセスは，その例外だからであり，メモリ・サイクルというMPUの内部機能を補助する機構へのアクセスとして適切な為でもある。従って，PMMUのレジスタはMPUの内蔵レジスタとしてソフトウェア上は扱うことができる。

以下に，PMMUの概略を紹介する。その詳細については，第12章を参照いただきたい。

●コ・プロセッサとなっている（cpID＝0はこのMMU専用)。
●自身にDMA機能を持っていて，外部メモリ中のデータ・テーブルを内部のレジスタにセットできる。
●CALLM, RTM命令をサポートする。
●BREAK命令をサポートする。
●68020と同じバス・インターフェースを持つ。
●アクセス・レベルのコントロール機能による高度の保護機能を有する。
●変換用のデータをツリー構造で，物理空間に配置

する。

次に，通常のMMUとしての諸特性を示す。

- ●1ページのサイズは256～32Kバイト可変となっている。
- ●チップ上には，変換用のレジスタを64エントリ持っている。
- ●ほとんどの場合，1クロック内で変換を終了する。

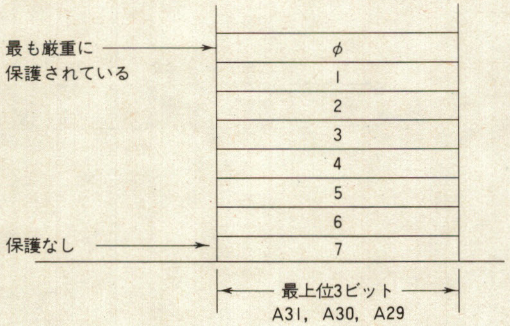
11-12-1 68851の概要

このMMCはデマンド・ページMMUと呼ばれるMMUで，論理アドレスと物理アドレスの変換を一定の上位アドレスに限って行ない，この上位アドレスには原理的には上限のないエントリ数だけ持てるというものである。この数多い論理アドレスと物理アドレスの変換値及び制御用のデータは物理アドレス空間中の実メモリ上に配置される。

この時，この変換用のデータはある数を単位として扱われ，各々のデータにディスクリプタと呼ばれる制御コードが付く。そして，このデータは，ツリー構造をとり，テーブル・データが次のテーブル・データを指し……と言う様に階層構造をとる事が可能である。

最終のデータか，次のためのポインタかは，ディスクリプタによりPMMUが認識する。そして，このメモリ中のデータは，ある論理アドレスから変換されるべき物理アドレス値をデータとして持ち，論理アドレス値はそのデータへ行くまでのインデックス値により計算されるべきものとして，これは持たない。各々のデータの構造には何種類かがあって「変換を行なわない」というものも一つの形式のデータである。

図11-12-1 アクセス・レベル・コントロールの構造



これらのデータは使用時はPMMUのＤＭＡ機能によりPMMU内の変換用キャッシュ・レジスタ（64エントリ）のどれかに取入れられて用いられる。そして，使われない，または，古くなったものは捨てられてゆく。この変換用のデータには，各種の制御コードが付けられており，これを通じて，キャッシュのアップ・デイトを禁止したり，バス・エラーを起こさせたり，書込みを禁止したりする。さらに，このPMMUは，階層構造を持つアクセス・レベル・コントロールという保護を行なう事もできる。

その他，これは直接にはMMUの機能とは関係はないが，ブレーク・ポイント命令という命令に必要とされるハードウェアのサポートを行なっている。これは，この命令をプログラム中にMPUが見つけると，ＣＰＵ空間の特定エリアをアクセスするのでこの時，代替えＯＰコードを返してやると，あたかもその命令をフェッチしたかの様に実行し，バス・エラーを返すと，イリーガル命令の例外処理を起すという命令で(詳しくは第７章７項参照)，この上記の作業を行なわせ，プログラムのデバッグを容易にする事ができる。

第12章

MMUの役割と機能解説

12-1　MMUとは何か

12-1-1　MMUの役割り

メモリ・マネジメント・ユニットを略してMMUと呼ぶが，この機能は高級なＣＰＵのシステムには欠かせないものとなっている。特に仮想記憶を使うシステムでは，必要不可欠のものとなっている。この主たる機能を以下に示す。

1)．論理アドレスと物理アドレスの変換を行なう。
2)．論理アドレスに対し，物理アドレスの拡張を行なう。
3)．メモリ・ブロックに属性を与え，属性により使用目的を限定する。

以上の内，1)の機能がなければMMUとは言わない。これは，ＭＰＵの出力するアドレス（論理アドレス）を実際のメモリ・アクセスに用いる物理アドレスに変換する作業である。この作業の意味する内容は幾つかあるが，その目的はソフトウェアの作成上の容易さと，実行速度の向上である。

まず，ソフトウェアの方であるが，現在では，システムの応用，実用のソフトウェアはほとんど何らかのコンパイラを用いて作成される。コンパイラ用のプログラムを記述する際にプログラムは，まず，メモリの物理上のアドレスは考慮する事はない。あるとすれば特別の場合の入出力関係のアドレスのみであろう。しかし，コンパイラとそれに続くリンカは実行用のコードを作り出す際，何らかのメモリ上のアドレスを必要とする。まず，プログラムをコンパイルする事により，ソースとして文字で記述されたプログラムをコードに変換する。

次にリンカによってコンパイラに登録されている関数，機能を必要に応じて（コンパイル時にリンカのために何が必要かの外部定義の一覧表は作成されている）接続し，全体に通しのアドレスを振る。この時，一般に，スタート番地，つまり通し番号の始まりは「＄０」である。これはプログラム・コード部も，ワーク・エリア部も同様である。そこで実際に様々なプログラムが作成され，そのスタート番地が全て＄０では，もし，そこの部分が使用可能であっても一時に一つのプログラムしか実行できなくなってしまうので（実際には不可能）個々に重複しないアドレスを与える必要が出る（これも難事）。

これに対する対処法がリンカによって，オブジェクト・コード（実行用のコード）に絶対番地を振り付けてしまうという低位のコンピュータで用いられる手法以外に二つあって，高級ＭＰＵと呼ばれるものならこの双方のどちらでも使える。

その一つは，プログラム・コード内には，Ｉ／Ｏアクセス以外の特定のアドレスの使用を禁じ（絶対番地を使わずに），コード全体をＰＣとの関係で定義するＰＣ相対で作成し，かつワーク・エリアはスタックの中にとる事にする方式である。こうすると，プログラム・コードはアドレス空間中のどこに置いても実行できるので，コード自体を作成する時にリンカの与える絶対番地は無意味になり，その始まりが＄０でも＄1000でも同じになる。

ローカル・ワークは全てスタック中にプログラム自体が作るので，ＯＳ，システムは特定アドレスのＲＡＭを与えなくともよい。これならプログラム・コードを実メモリ中に置く位置の制限がないので，システムはメモリを有効に使うべく，コードの置く位置を任意に指定し，適当なスタックを与える事により実行させる事ができる。

この方法には，二つの欠点がある。その一は，メモリ中には幾つものプログラムが同時に存在し，スタックも同居する。そこで，あるプログラムのバグまたはデータ異常による誤動作はシステムの他の部分へ干渉してしまう事が考えられる。もちろん，自身のプログラム・コードを意図外に変更してしまう可能性も大きい。こういう事態が起きるとシステム全体が暴走してクラッシュする事も多い。

もう一つの欠点は，一般にＰＣ相対のアドレス・モードを使用したプログラム・コードは遅い。つまり実行速度が遅い事がある。これは，コードに含まれるＰＣ相対の数だけアドレス計算を余分にしなければならないためである。

この様な欠点をとり除くのが対処法の二つめであるMMUを用いたシステムである。

この場合はプログラム・コードは，その最も効率

の良い絶対番地を用いて記述されて良い（もちろん相対の方が効率の良い場合は相対で)。ワークも何ら制限なく設けて良い。これらのプログラム・エリア，データ・エリア，他のプログラムと共用するコモン・エリアはMMUにより，それぞれが必要とする値を与えられる。従って＄0であろうと＄1000であろうと，まったく別のアドレスの実メモリに割当てるので，この時，変換に用いるためのオフセット値が異なるのみである（このオフセット値は論理アドレスと物理アドレスの差値で，MMU内のレジスタに影響するのみである)。またMMUはある一つのプログラム用に設定されている時は，実メモリの他の部分が使用されない様に保護を行なうので，ソフトウェアのバグや異常なデータ等による予期されない動作に対してもシステムの安全性は高まる。

ただしこのMMUを使ったシステムでは，ＯＳのソフトウェアはかなり大きく，重くなる。更にシステム全体としてもMMUを使う事を前提で組上げられていなければならない。これは，まずＯＳ側において，MMUのイニシャライズと実行タスクを切替るごとに，MMU内のタスク番号のレジスタを変更してやる必要がある他，MMU内のレジスタにあるタスクを実記憶にロードする時に，同時にそのプログラムに伴う情報エリアより，プログラムのアドレス範囲，スタート・アドレス，ワーク・エリアのアドレス範囲を知って各オフセットを計算し，これに保護属性を付加したものをセットしなければならない。そして，このタスクにタスク番号を与えてグループとしてアクティブになる様にし，この番号をＯＳ内のタスク・テーブルに登録する。

さらに，このMMUが用いる変換用のツリーを作成して，ルート・ポインタの指定により，これを正しく取込める様にする必要がある。

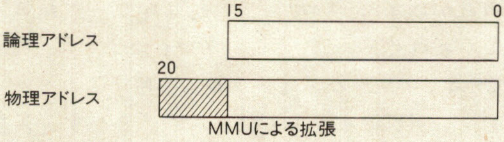
MMUのレジスタが不足すれば（つまり変換できるチャンネル数が足りなくなると)，ＯＳはどれか使用頻度の低いタスクのレジスタにセットされているのを解除して，レジスタ値のコピーを保存して，他のタスクに割り振る事も行なう必要がある。特にこれらの作業のあるものは物理的なタイミングを要求する場合もありえるので，ＯＳのMMUドライバは注意深く作成する必要がある。MMUは，ハードウェアとしてもＭＰＵの実行を補助する部分も必要である。それは，ハードウェアからの割込みや，ソフトウェア割込みによりタスクが中断され，ＯＳ側(スーパーバイザ・モード）に移行した時，その最初のスタック・アクセスや，ベクタ・アクセスに対しＯＳ側の空間のアクセスを保証する必要があるので，ＭＰＵのハードウェアと共にこの切替えを自動的に行なう必要がある。さらにMMUには幾つかの欠点がある。その一つは，ＭＰＵが出力してくるアドレス（論理アドレス）を何らかの方法で実メモリのアドレス（物理アドレス）に変換するために時間が必要だという事である。この時間は，各MMUの変換方式によって異なるが，等価的にはメモリのアクセス・スピードが落ちて，メモリ・アクセスに時間がかかる事に等しい。そこでＭＰＵの処理速度が，かなりの部分をメモリに依存しているシステムではＭＭＵのためにハード的に実行速度が落ちる。これを補うために，実行時間の短いアドレス・モード，プログラム構造をとって効率を上げようとする。これは，MMUのあるシステムでは，特にＰＣ相対等の複雑なアドレス・モードを使う必要性は少ないからである。

12-2　MMUのその他の機能

これは68000系のMMUでは使われていないが，8ビットＭＰＵや低位の16ビットＭＰＵでは，ＭＰＵのアドレス空間を16ビット分しか持っていないものが，ほとんどである(例　6809，8086)。ところがこれらのＭＰＵでも時として，数100Ｋバイト以上のデータを扱ったり，ＯＳそのものが16ビットのごく狭いアドレス・レンジを超えてしまう事がある。この様な時，システムはMMUを採用して，アドレス空間を拡げる事ができる。これは先のオフセット値を上位に拡張しておけば良く，このオフセット・レジスタの切替えで上位のアドレスに対処する（一般にこの切替えを自動で，または切替え要求をＭＰＵに割込んで行なうなど)。現在の半導体技術は，高位の16ビットＭＰＵに対し，このMMUを同一チップ上

に集積する事を可能としているので，かなりの高級ＭＰＵといわれるものがMMUを実装している（例68030，68070，Ｔ88000，i80186，など。図12-2-1)。

図12-2-1 MMUによるアドレス拡張



12-3 ＭＭＵの種類

MMUには大きく分けて，ページ・モードとセグメント・モードの２種類がある。以下に，これらの概要を説明する。

12-3-1 ページ・モードとセグメント・モードの基本的な差異

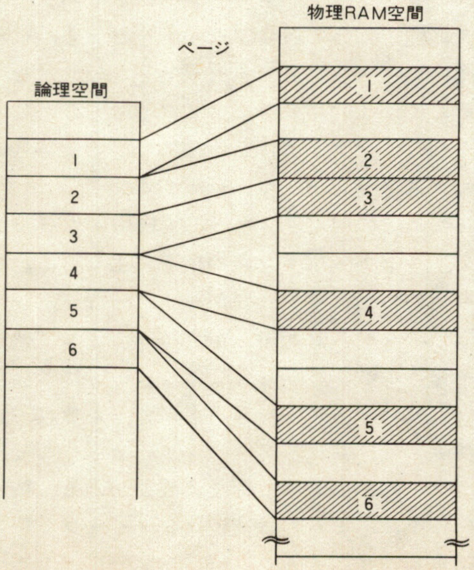
この両者の差異はメモリ空間の分割管理を行なう上での処理方法の違いである。ページ・モードの場合は，実際にＲＡＭが組込まれているメモリ空間を比較的小さい「ページ」という単位長に分割し（通常２Ｋ～４Ｋが多い)，このＲＡＭの空間全体をページという一定長のメモリ空間の集合であると考えて行なう処理方法である。つまり，１Mバイトのワード幅が16ビットのＲＡＭ空間があれば，これを４Ｋバイトのページに分割すると，256ページに分割されこのページ単位にて管理される。つまり85Ｋバイトのプログラムがあれば，このプログラムには85／４＝21.25→22ページ分のＲＡＭ空間を与える事になる。そして最後のページでは，４Ｋバイト中，１Ｋバイトしか使われず，３Ｋバイトは余りとなり使用不可となる(図12-3-1)。

もちろん，この22ページのメモリ空間は物理ＲＡＭ空間中にまとまって存在しても良いし，散在していても良い。論理アドレスと物理アドレスの変換は各ページごとに個々に行なわれるので，この対応はいずれの場合でも自由に行なえる。

セグメント・モードの場合は，物理ＲＡＭ空間の分割法が異なり，「セグメント」という不定長のブロックに割当てられる。プログラムは，プログラム・コード自体であるプログラム・セグメントと，データを入れるデータ・セグメント，共通領域となるコモン・セグメントといった用途分類により分割されこの機能セグメントを，一つまたは幾つかのメモリ・セグメントで構成する。ここで扱うセグメントは分類法ではなく，メモリのセグメントである。

このメモリ・セグメントは各セグメントごとに大きさが異り，各プログラムとその機能分類ごとに，最低１個のメモリ・セグメントを持つ事になる。前例と同様に，85Ｋバイトのプログラムがあると，これを，64Ｋバイト，16Ｋバイト，８Ｋバイトの三つのメモリ・セグメントで構成する。この時，８Ｋバイトのセグメントのうち，３Ｋバイトは余りとなり使用不可である。この時，物理メモリに連続して64Ｋバイトの空間が取れなければ，この64Ｋバイトのセグメントは２個の32Ｋバイトのセグメントで構成しても良いし，〔32Ｋ＋16Ｋ＋16Ｋ〕の様にしても良い。もちろん，この順序はどの様であっても良い。物理アドレスと論理アドレスの変更は各セグメントごとに行なわれるので，各物理メモリ・セグメントは集中していても良いし，分散していても良い。この

図12-3-1 PMMUによるアドレス変換の実際



方式の場合はＲＡＭ空間の大きさが定まっていても幾つのセグメントが使用可能となるかは予想できない。システムで使用可能な最大メモリ・セグメント数は，ＭＭＵ内のセグメント・チャンネルの数によって制限される事になり，これを越えて使うのはソフトウェアによる入替えのサポートを必要とする。

図12-3-2 セグメント・モード・MMUによる変換

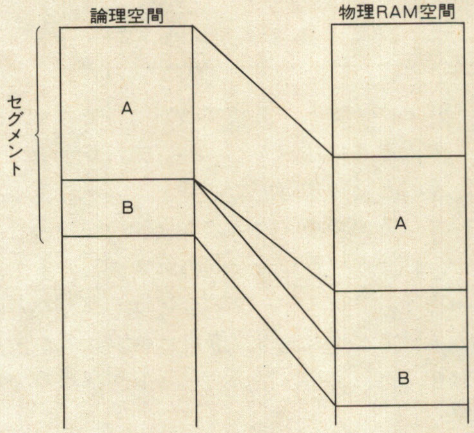


図12-3-3 メモリ空間の不利用部分の発生(1)

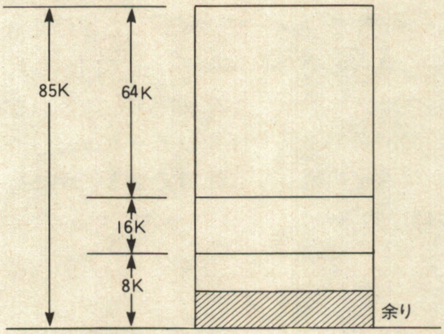
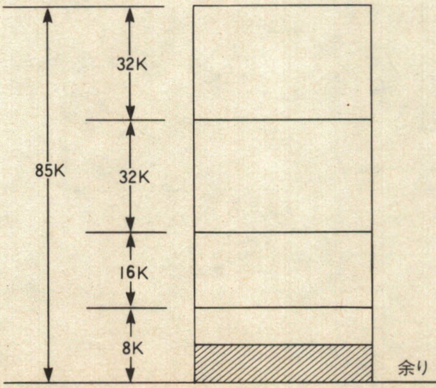


図12-3-4 メモリ空間の不利用部分の発生(2)



12-4　MMUによるアドレスの変換法

物理アドレスと論理アドレスの変換にも幾つかの方法があるが，現在，よく用いられるのは，メモリによる入替え法である。ただし小規模なシステムでは，加算法によるMMUも残っている。

入替え型のMMUは，現在の主流を占める方式であって，**図12-4-1**の様な構成をとる。デコーダ部のメモリには論理アドレスが入れられ（不要のビットはマスクする様に設定する)，ＭＰＵが出力してくる論理アドレスと比較をして，一致した場合に，レジスタ・メモリ部の内容を物理アドレスとして出力する。そこで，予め，レジスタ・メモリには変換されるべき物理アドレスを必要ビット数入れておく。下位のアドレスと先にマスクしたアドレスはそのまま論理アドレスが物理アドレスとして出力される。

この方式の利点は，構成が簡単な事，デコーダ部では一致のみをみるので比較的単純で高速，レジスタ・メモリ部は単なるメモリ・アクセスなので，現在の半導体技術上，高速に行なえる事と，全体に高速化，単純化が容易で判りやすい事があげられる。

加算型のMMUは現在余り用いられないが，従来比較的小さいシステムでよく用いられた。これは**図12-4-2**の様な構成で，論理アドレスの使用空間の上限と下限を限定して，この中に入っている場合に，論理アドレスにオフセット値を加算して物理アドレスを得るものである。

この方式は，上限，下限で限定する空間は不定長となるのでセグメント・タイプの一方式となるが，

図12-4-1 入れ替え型・MMUの構成

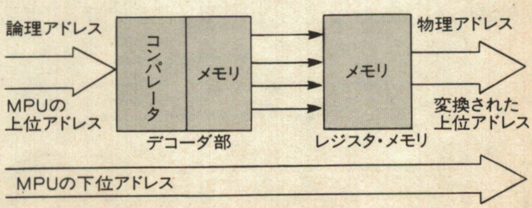
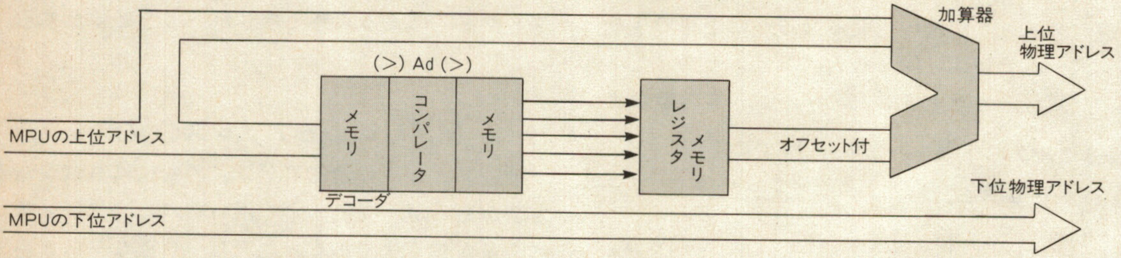


図12-4-2 加算型・MMUの構成



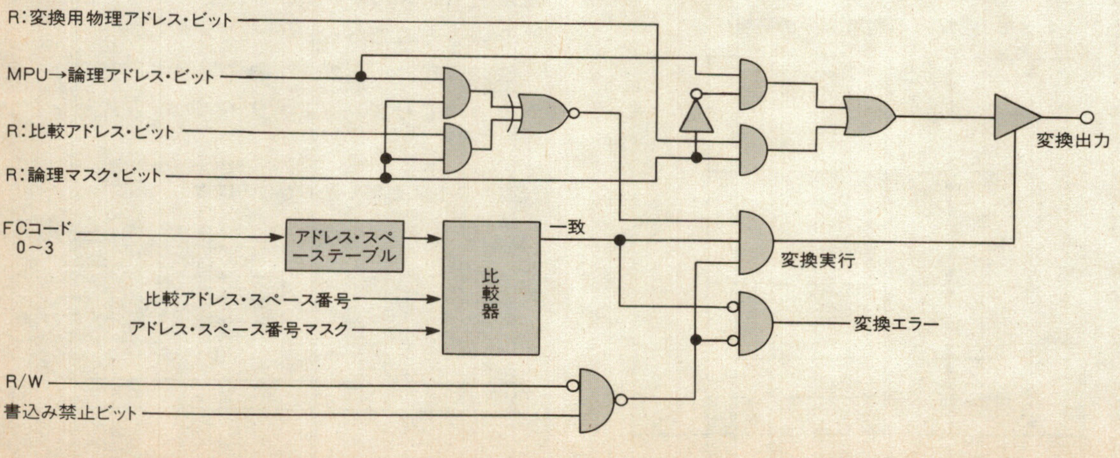
メモリ空間をかなり細かく限定できる所にあり，先の85Kバイトなら，一つのセグメントでこれを実現してしまう。つまり，セグメント分割の最小単位まで，半端な空間長をサポートできる。ただし，この方式は，必要とする空間を連続でとれる時に最高の効率を発揮し，細かいセグメントを接ぎ合せると，効率は悪化する。それは，この方式では，デコーダ部にウィンド・コンパレータといって，二つのコンパレータと，その大小判定の論理回路を必要とする（イコールのみの場合は，EXCLUSIVE－ORのみで一致がとれるが，大小判定は，複雑な論理を必要とする)。さらに，レジスタ・メモリから呼出した値と加算を行なう(キャリー・アップに時間がかかる。加算器も必要)。上記の様なハードウェアおよび実行時間を余分に必要とするからである。

12-5 MC68451

このMMUは68000／10用のMMUである。68851とは異り，セグメント・タイプのMMUであり，セグメントの大きさは256バイトから16Mバイトまで，各チャネルごとに設定可能であり，このチャネルを32個持っている。変換メカニズムは，比較用のビット比較を，行なう／行なわないのマスクビット，置換え用の物理アドレスデータの３ビットからなるが各チャネルにアドレス24本，ＦＣコード４本が用意されており，更に書込み禁止のビットを持つ（図12-5-1参照)。

この場合，ＦＣコードから導かれるタスク番号が一致するチャネルが開かれ，各タスクのプログラムセグメントやデータ・セグメントごとに変換を実行する。これら必要なレジスタは全て内蔵され，メモリ中にデータ・テーブルを展開することはない。またこのMMUは１個では32のチャネルしか持たないので，複数を接続しても使用できるようになっている(詳細は個別のデータ・シートによる)。

図12-5-1 PMMUのアドレス各ビットによる処理方法のイメージ



12-6　PMMU68851

PMMU68851は内部に大きく分けて二つの機能ブロックを持っているコ・プロセッサである。その機能１はＭＰＵの出力してくる，つまりソフトウェアのアドレスを物理メモリのアドレスへと変換するＭＭＵの機能であり，その機能２は，BKPT命令をサポートする外付けハードウェアとしての機能である。

この章では，比較的簡単な機能であるBKPT命令のサポート機能は簡単な紹介にとどめ，アドレスの変換機能，つまりMMUの機能に重点を置いて解説を行なう。

12-6-1　PMMUの意義

68851は，マルチ・タスク，マルチ・ユーザーのＯＳをサポートする為のMMUとして開発された。

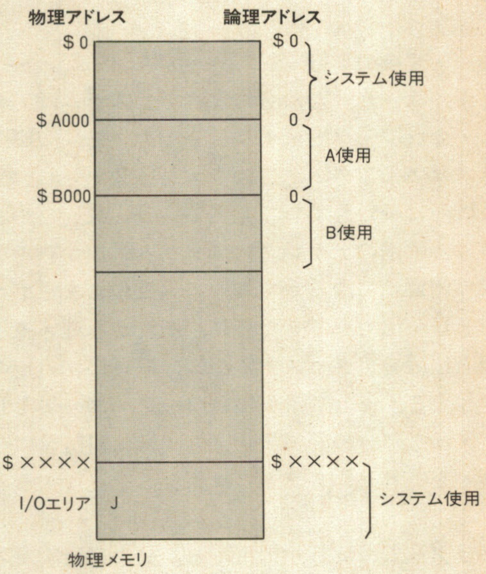
それは，この様なシステムでは，他のユーザーのシステムの使用部分の保護，ＯＳ自体の保護，仮想マシンの作り易さなどが主題となってきて，単にポジション・インディペンドのオブジェクト・コードを用いてのマルチ・タスク化で不充分となってきたからである。

その意味で，このPMMUは，マルチ・タスクの環境サポートも行なうし，大規模のシステムでは必ずといっていい程，採用されるキャッシュ・メモリの外付けにも配慮が行なわれている。

12-6-2　アドレス変換方法とアドレス変換テーブル

PMMUは，ＭＰＵの出力してくるアドレスと，物理メモリへのアクセス・アドレスを変換する。これが，まずどんな場合に必要になるのであろうか？容易な例として，あるシステムを２人のソフトウェア・エンジニアが使っており，２人共，同じアセンブラを使い始めたとしよう。仮に，この２人をＡ，Ｂとすると，Ａはアセンブラのプログラム自体と自分専用のワーク・エリアを必要とするし，Ｂも同じ様にアセンブラのプログラム自体と自分専用のワーク・エリアを必要とする。特別な場合には，アセンブラ・プログラムは１個で済むＯＳもあるが（ＯＳ-９），ここでは無視する。

図12-6-1　同時に2人が同じ論理アドレスを使用



この場合，アセンブラ自体がシステム・メモリ上に二つ存在し，アセンブラの使用するワーク・エリアも二つ存在する。ＭＰＵはＯＳによって，このＡ，Ｂの二つのアセンブラをタイムシェアリングによって等しく実行するとする。

当然，二つのアセンブラの実行モジュールのアドレスは同じであり，二つのワーク・エリアのとられ方もほとんどの場合，同じであろう。

ところが，このアセンブラ二つとワーク・エリア二つは，同じメモリ空間中に置かれなければならないから，当然，本来のプログラムの実行用のアドレスとは無関係なメモリ空間の空きエリアに入れられる事となる事となる（図12-6-1参照）。

その結果，もしアドレス変換がないと，ＭＰＵはそのコードの置かれているアドレスと実行用アドレスの関係に違いを生じてしまう。PMMUは，ここでアセンブラの実行が正しく行なえる様なＭＰＵの使用するアドレスを，メモリ空間中に任意の位置に置かれたプログラム／データに対して，その物理アドレスとの関係を調節して与える事ができる。

実際的には，ＭＰＵの出力してくるアドレスをメ

モリ空間の任意のアドレスに変換できれば，この関係の調整はとれる事になる。

12-6-3　変換の実行

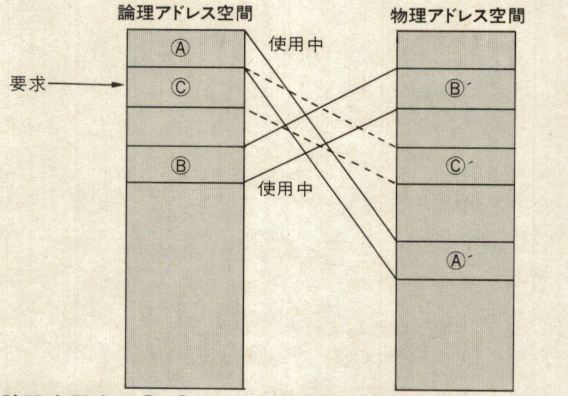
まず，ユーザーの命令によってシステムはアセンブラ・オブジェクト・モジュールをメモリ空間中の空きエリアに置き，そこの番地を確認する(この時，複数個所に分かれていても良い)。そして，このプログラムの本来のアドレスのスタート番地が物理アドレスの何番地になるかを調べて，このスタート番地をMPUが出力してきた時に，それが物理メモリ上の相当位置をアクセスする様にMMUに変換用データを与える。そして最小限のワーク・エリアとしてのスタック部分の変換データも同様にMMUに与えれば，先のアセンブラ・プログラムを実行する事ができる。

プログラムの実行が始まると，MPUは次々とプログラムの他の部分や，データを作ろうとする部分をアクセスする。そこでMMU内に変換用のデータがないとMMUはバス・エラーを発生する。MPUは，このバス・エラーを受けて，その内容をチェックして，MMUの変換用データの不足と判断すると新たに変換用のデータを追加していく。このMPUの実行に伴って，要求があると(デマンドがあると)その部分を用意してゆくというのが68851というデマンド・ページMMUの使用コンセプトである(図12-6-2)。

12-6-4　ページ・サイズ

さて，各変換を行なうのであるが，1MPUアドレスを1物理アドレスに変換していたのでは1対1対応のメモリが必要で，とても物理的に不可能であるし，非効率である。又，プログラムにしてもデータにしても，必ずある最低限の大きさを有すると考えられる。そこで，下位の何ビットかはMPUの出力してくるアドレスをそのまま物理メモリのアドレスとして用いる事が考えられた。このPMMUでは，A0～A7は，PMMUのアクセスには用いられても変換される事はない。つまり，変換はA8以上の予め指定される上位のアドレスのみに行なわれる(図12-6-3)。

図12-6-2　MMUの使い方のコンセプト



論理空間中のⒶ，Ⓑは物理空間のⒶ′，Ⓑ′に変換されている。今，要求がⒸに対してあると，バス・エラーが生じ，MPUは新たな変換データを作ってⒸを物理空間の何処かにⒸ′変換できる様にする。

図12-6-3　ページ・アドレスの変換

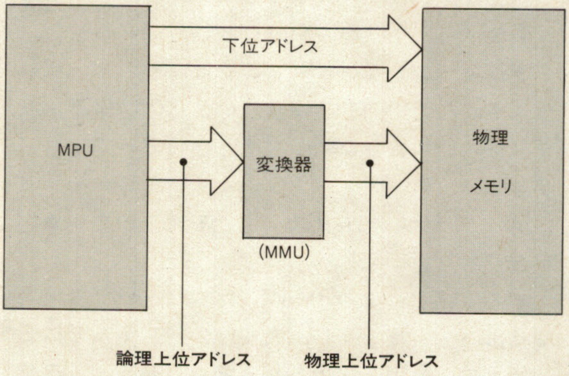


図12-6-4　TCR内のページ・サイズ・コード

$8 — 256 Bytes	$C — 4K Bytes
$9 — 512 Bytes	$D — 8K Bytes
$A — 1K Bytes	$E — 16K Bytes
$B — 2K Bytes	$F — 32K Bytes

この変換され得ない物理メモリ空間の大きさがページと呼ばれるものである。A0～A7が変換されなければ，1ページはこのA0～A7の8本で指定される256バイトである。

このページの範囲は，PMMU内のTCRというレジスタの該当するフィールドをセットする事により指定でき，$8～$Fのコードにより256バイトから32Kバイトまでを指定できる。64個の変換データを

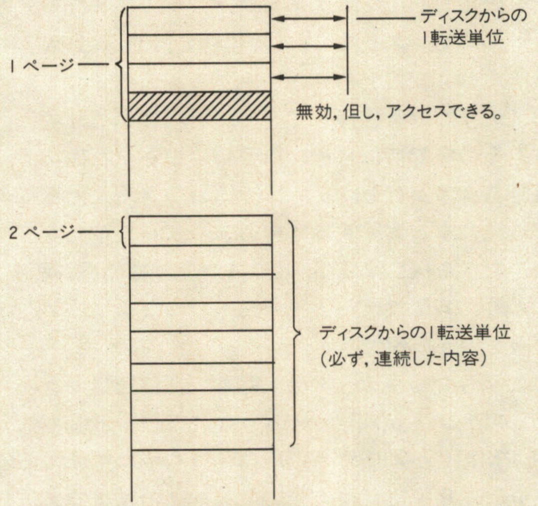
PMMUは内蔵できるので，重なりがなければ最低は32Kバイトから最大2Mバイトのメモリ空間の変換をPMMUはその内蔵データのみで実行できる事になる（図12-6-4）。

このページのサイズは，外部記憶のハード・ディスクの1データ・ブロックの整数倍の大きさが一般には用いられる。これはディスクとの入出力の単位は256バイトであったり，1Kバイトであったりするが，このブロック・サイズ以下のPMMUの変換は，例え行なうとしても常々ある最低の連続したブロックに対して行う事になるので，単にPMMU内の変換データ・エリアを使いつぶしているに過ぎなくなる。この様な連続したブロックは，下位アドレスの変換の必要がないので，その大きさとページの大きさを合わせてしまって良い。

逆に，ディスクから3ブロック読んできたところこれを1ページに割り当てる時に，4ブロックの大きさが必要な場合は，最後の1ブロック分はPMMUの細かな保護を与えられない。そこでやはりあまり大きなディスク側のブロック・サイズとPMMUのページ・サイズの差は好ましくない（図12-6-5参照）。

PMMUはアセンブラ・プログラムの様な大きなブロックに対しては，物理メモリ上に置く変換データのテーブルを簡略化する方法はあるので，最低ページ・サイズは，やはりディスク側との小ブロックのやりとりを基本とすべきであろう。

図12-6-5　ページの大きさとディスクのセクタ・サイズ



12-6-5　タスク・エリア

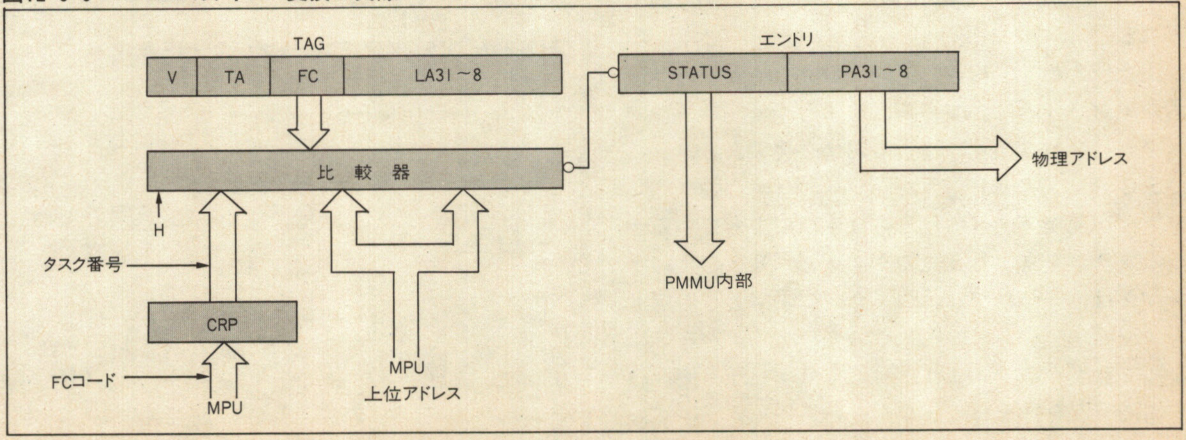
さて先の例に戻るが，前述の様にしてAとBの二つのアセンブラの為にプログラム用，データ用の変換テーブルが作られて交互に実行されているとする。そこで，もしユーザーAのアセンブラの置かれている物理メモリの番地を＄A000，ユーザーBのアセンブラの置かれている物理メモリの番地を＄B000とする。そしてアセンブラ自体の番地は0であるとしよう。すると次の変換データがPMMUの内にある事になる（図12-6-1）。

ユーザーA：＄000→＄A000（U.P）
ユーザーB：＄000→＄B000（U.P）

後段の（　）内は，FCコードの内容である。

つまり，MPUが，ユーザー・プログラムとして

図12-6-6　PMMUのアドレス変換の実際



の＄０番地をアクセスに行くと，二つの変換データがある事になる。これでは，どちらの変換データを用いて良いのか判らなくなってしまう。

この問題には，二つの解消法が考えられている。その１は，ユーザーの切替時には，PMMU内の変換データを全部消してしまい，新たに新しいユーザーのデータを読み込ませる方法であり，その２は，ユーザーごとに（又は，タスクごとに）タスク番号を付けて，このタスク番号の一致している変換データのみを有効として用いる方法である。PMMUは，いずれの方法もとれるが，主に，２番目のタスク番号による方法を主とする。

よりサイズの小さいPMMUの場合には，１の方法しかとれないものがある(68030のMMU)。

この機能は，PMMU内のＣＰＵルート・ポインタという変換データのテーブルを指定するポインタに付属するタスク・エリアというコードによって成される。このタスク・エリアは３ビットのコードであり，PMMU内の各変換データにＭＰＵのアドレスの一部として付けられている。そこで，このPMMUは，同時に８個まで並列に働くタスクの変換データを内蔵できる。この８個の各タスクに対し，プログラム／データの双方が持てるのはいうまでもない。

このタスク・エリアは，新しい変換テーブルをＣＰＵルート・ポインタに与えると空きがあれば，自動的に割り当てられ，空きがなければ最も古い，つまり，最も使われていないものが消され（そのタスク・エリアに伴う変換データも同時に消され，変換データ・メモリの空きも作られる），そこに割り当てられる。

注意点として，複数のタスクが共有するメモリ・エリアがＯＳでは必ず必要になるし，ユーザー・プログラム間でも相互の通信の為に必要である。そこでＳＧ（シェアード・グローバル＝全体で共有する）フラグが変換データに付けられる様になっているがこのフラグを立てると先のタスク・エリアの比較チェックは行なわれない。つまり一つのＭＰＵのアドレス（論理アドレス）に二つの変換データが有効になってしまうので，これに対してはユーザー側，ＯＳ側で管理を行なわねばならない。

これ以外にも，先のアセンブラの例でデータ領域も又＄０番地から始まっていたらどうなるのであろう。これはPMMU内でＦＣコードが有効となっていれば，プログラムとデータは分けられるので問題はないが，このＦＣコードの入力の禁止も内部レジスタでできるので，禁止する場合には注意を払う。

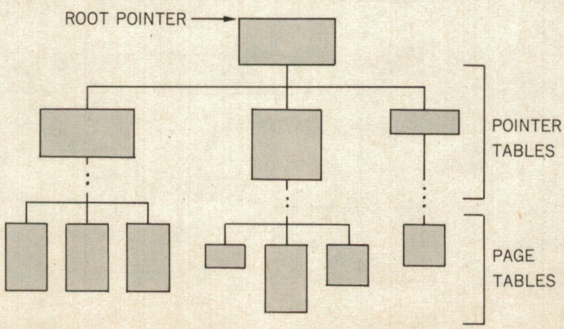
更に，各変換データにはシステムの保護の為の幾つかのフラグが付けられている。例えば先のアセンブラ・プログラム自体の変換データにはＷＰ：ライト・プロテクトを立ててプログラム自体に何があっても書込まない様にするべきであろう。

12-6-6 PMMUが動作している環境

PMMUはRESETスタート時には何もしない。ＭＰＵの出力してくるアドレスは，そのまま物理メモリへのアドレスとなる。この状態でＭＰＵはPMMUへ渡すべき変換データのテーブルを入れておくためのメモリ部分を確保して，これを物理メモリ中に作り，その内で特に自己のＯＳ用のデータを作って，自分自身の実行と，テーブルへのアクセスに支障のない様にする事，例外処理のテーブルも正しくアクセスできる様にする事，といった環境整備を行なう。この時，各変換データをチェックする為にPTEST命令を使う事ができる。

これらの変換データは，実際には「論理アドレス＝物理アドレス」の場合が多い。これはＯＳの基本部分や，PMMUが動作する以前にメモリ中にすでにある部分は，変換実行後も元のアドレスのままである必要がある。つまり「論理アドレス＝物理アドレス」と

図12-6-7　アドレス変換用テーブルのツリー構造



なる（これらの部分はある大きな部分が，連続していると思われるが，この様な変換データ・テーブルは簡略化して作成できる)。

そして，これらの基本プログラム部分で，特に多用される部分は一度，PMMU内に入れられると，その消去を禁止する事もできる。これが，ロック・（Ｌ）フラグである。タスクのディスパッチを行なう部分や，割込みハンドラ等ではこのフラグを立てておき応答の速さを保つ（Ｌフラグが立っていない場合には，消去され得て，この時，その新たな使用では物理メモリ中のテーブルをアクセスする必要と，その為の余分な時間を消費してしまう。

さて，PMMUが使う物理メモリはどの様な特質が必要であろうか？PMMUはそれ自体で内部の変換データの選択時にＦＣコードを用いる事も禁止する事もできるから，物理メモリはＦＣコードのデコードはされていても，いなくても良い。但し，一般にデータもプログラムも同じ物理メモリ上に置き，プログラムにはＷＰ(ライト・プロテクト)を行なう場合が多く，この時，メモリ側ではＦＣコードのデコードは行なわず，ＰＭＭＵに行なわせるのが普通である。もちろんＦＣコードにより物理メモリを分けていても良い。この場合には，通常の変換時，ＦＣコードの変換はなされないので従来通りの分割もできるし，又は，各々独立した空間を作る事もできる。この様に物理メモリ側でＦＣコードをサポートした時は，通常のＭＰＵのアクセスでは，PMMU内のＦＣコードの処理を禁止してもできる。

12-6-7　ＦＣコードとの関係

PMMUには，ＦＣコード入力は４本ある。ＦＣ０，１，２はＭＰＵとＤＭＡから，そしてＦＣ３のＨはDMACからと決められている。通常の変換時には，ＦＣコードは他の論理アドレスと同様に変換の為の入力となっているが，一つだけ例外があって，ＦＣ＝＄７の時（４ビット・コード）つまりＣＰＵスペースをアクセスしている時は変換を行なわずに論理アドレスを，そのまま物理アドレスとして出力する。これはＣＰＵスペースの各機能は，アドレス変換を行なう事は不要であり，無用の混乱をまねく怖れがあるからである。事実ＣＰＵスペースへのアクセスで，一般の物理メモリへのアクセスはあり得ない（更に例外として，アドレスの変換も出力も行なわない場合がある。それはPMMU自体へのコ・プロセッサとしてのアクセスである)。

これによりFPCP等の他のコ・プロセッサはPMMUの外に付けても，内に付けても良くなり，割込みアクノレッジ・サイクルも従来通りに実行される。

PMMUがＦＣコードを出力する場合が一つだけある。それは物理メモリ中に置かれた変換データ・テーブルをアクセスする時である。この時，PMMUはバス・マスタとなって自己のみでメモリ・サイクルを実行するが，これに伴ってＦＣ＝＄５（スーパーバイザ・データ）のＦＣコードを出力する。これは例外的であっても，物理メモリ側で，ＦＣコードをデコードしている場合に対応したものである。これは先の物理メモリ側にてＦＣコードをサポートしているシステムに対応する。

更にＦＣコードは，PMMU内の変換の為のデータの選択（先の例のタスク番号）に用いられる。これは，ＤＭＡ時とスーパーバイザ時，ユーザー時の３分類をPMMU内で行なっているからである。但しスーパーバイザ時とユーザー時は同一化できる。そして，ユーザー時には，そのユーザー・タスクの番号を変換データの選択に用いる。

12-6-8　変換の実際とＡＴＣ

PMMUのアドレス変換は，68020と組合わせた時には，ＭＰＵクロックと同一のクロックを供給して１クロック以内に終了し，ＭＰＵのＡＳに対して１サイクル遅れのPMMUからのＡＳ＝ＰＡＳが出力されてくる。ＭＰＵと異なったクロック源を用いた場合や，他のＭＰＵ(たとえば68010)と組合わせた場合，また，DMACがバス・マスタとなった場合にはSYNC入力をアサートして，各アドレス入力ＭＰＵのＡＳ（ＬＡＳ）で同期をとってから変換する。この場合には変換に２クロック・サイクルを要する。

ここでは，主な用途である68020と組合わせた場合を想定して解説を進める。

ＭＰＵがメモリ・サイクルをスタートさせ，アドレス（論理アドレス），R/W，ＦＣコードを出力してくると，PMMUはその内部でこの各アドレスと，それに附随してくる信号とをPMMU内部に持っている変換データに相当するものがあるか否かを捜しにかかる。これらの変換データは，ＡＴＣ（アドレス・トランスレイション・キャッシュ）と呼ばれる。入力アドレスの比較器を個々に有するキャッシュ・メモリに蓄積されている。このキャッシュ・メモリは68020の命令用のキャッシュと同様にアクセスされる時の条件を示すＴＡＧという部分と出力する内容を示すエントリという部分の対でできており，68851では，これが64組内蔵されている（**図12-6-8，12-6-9**参照）。このＴＡＧには，命令用のキャッシュの時にはＳフラグ（スーパーバイザ），Ｖフラグ（有効）のフラグがあり，アクセスされた命令のアドレスが入れられていた。それと同様にPMMUのＡＴＣのＴＡＧには，Ｖフラグ（同様に「有効」），タスク・エリア・コード（前述），このタスク・エリアを無視すべきというＳＧフラグ，ＦＣコード，論理アドレスが納められている。エントリの方には命令用のキャッシュでは命令コードのみが納められていたが，PMMUのＡＴＣには，物理メモリに出力するべき物理アドレスが納められている。又，同様にシステムやデータの保護に関わるステータス・フラグが６ビット納められている。

ＭＰＵからのアクセスがきて，論理アドレス，ＦＣコードが与えられると，この内容は同時に64個のＴＡＧの内容と比較される。この比較は各ＴＡＧに組合わせられた比較器（EXORで一致をみる）が，これも従って64セット存在し，これらが同時に比較作業を行なう。これと同時に与えられたＦＣコードにより，ＣＰＵルート・ポインタ（ユーザー用のＦＣコードで使用されるポインタ）が使われる場合には，このＣＲＰ（ＣＰＵルート・ポインタ）からタスク・エリア・コードが出力されて，このコードも同様ＴＡＧとの比較に参加する。

そこで，予め指定された範囲に於て全ビットが一致すると，そのＡＴＣはＨＩＴした事になり（命令用キャッシュと同様）エントリの内容の出力を行なう。全64ＡＴＣにＨＩＴのない場合は後述するテーブル・サーチをPMMUは行なう。

図12-6-8　ATCタグ・エントリ

SG	TA	FC	LOGICAL ADDRESS

図12-6-9　ATCデータ・エントリ

L	B	C	W	M	G	PHYSICAL ADDRESS

ＨＩＴすると，そのエントリから読出されてまず用いられるのはプロテクション用のステータス・フラグ，６ビットである。この各ビットの内容はＰＡＳの出力タイミングまでに内部で使われ，ＰＡＳのタイミングと同時に有効となり出力される。その内容が物理メモリへのアクセスを許さない様なものの場合には，ＰＡＳは出力されない。

もし何の違反も生じなければ，エントリに含まれている物理アドレスの予め決められている部分と，Ｃビットが PMMUから出力される。物理アドレスはメモリに与えられるアドレスとして，Ｃビットはデータ・キャッシュ（外付）への書込みを禁止する信号としてである。そして物理メモリのアクセスを行ない，通常のＭＰＵと同じメモリ・サイクルを実行する。

このステータス・フラグには，以下のものがある。

1)．Ｌ（ロック）：このエントリは常に存在して置き換えられない事を示している。

2)．Ｂ（バス・エラー）：このエントリが呼び出されるとPMMUはバス・エラー信号を発生して，そのメモリ・サイクルを中断する。これはテーブル・サーチ中に，バス・エラーが発生したか，モジュール・コールで許されないアクセス・レベルの保護の為にセットされているかを示している。

3)．Ｃ（キャッシュ・インヒビット）：このビットの反転したレベルがPMMUの出力ピン$\overline{\text{ＣＬＩ}}$をドライブする。これは外付けのデータ・キャッシュへの書込みを禁止する為のものである。

4)．Ｗ（ライト・プロテクト）：このビットが立っている時には，このエントリへの書込みは禁止され，このエントリを呼び出したメモリ・サイクルがライ

ト・サイクルなら，このサイクルはバス・エラーで終了する。

5)．Ｍ(モディファイド)：このエントリには既に書込みのあった事を示している。一度セットされると自動的にクリヤされる事はない。

6)．Ｇ（ゲイト）：このビットは，このエントリにCALLM命令のモジュール・ディスクリプタを持てる事，つまりCALLM命令で呼び出され得る事を示している。

以上のビットは，物理メモリ上の変換データ・テーブル上の各変換データにも同じく与えられている。特にMビットは，PMMUがあるエントリへの書込みを認識すると，そのATC内のMビットをセットするのみでなく，このテーブル上の対応ビットをもセットして，ATCを消去しても，テーブルより読み出せば再生される。

12-6-9　変換データ・テーブル

さてPMMUには64個のＡＴＣが変換データを有している。そして，各ＡＴＣは，予め決められたページ・サイズのメモリ空間の変換を実行している。しかし，最大でも２Ｍバイトの変換を行なえるだけであるとも言える。もちろん，これだけでは不充分である。そこで何らかの予備の変換データが必要となる。

このPMMUは，変換データを全て，使われている，いないに関わらず，物理メモリ上にツリー構造によってアクセスされるテーブルとして持ち，このテーブルをPMMUはＭＰＵの補助なしにアクセスしてテーブルに納められている変換データを自己のＡＴＣのエントリに取込む事ができる。

実際には，ＭＰＵはPMMUの使用前にこの変換テーブル骨組みを作っておきソフトウェアの要求(デマンド）によってメンテナンスを行ない，変換テーブルを充実させていく。つまり，まず変換テーブルに変換データを作り，これをPMMUが読んでゆくという形になる。もちろん，この変換データには，ステータス・フラグの内容や，ＴＡＧのＳＧビット，CALLMの階層指定等の情報も含まれる。このＡＴＣに入れられる変換データそのものを記述している

図12-6-10　TYPE1, TYPE2ショート・フォーマット・ページ・ディスクリプタ

31–16							
PAGE ADDRESS (PA31-PA16)							
15–8	**7**	**6**	**5**	**4**	**3**	**2**	**1–0**
PAGE ADDRESS (PA15-PA8)	G	CI	L	M	U	WP	DT

図12-6-11　TYPE1ロング・フォーマット・ページ・ディスクリプタ

63–48										
UNUSED										
RAL	WAL	SG	S	G	CI	L	M	U	WP	DT
PAGE ADDRESS (PA31-PA16)										
PAGE ADDRESS (PA15-PA8) (15–8)	UNUSED (–0)									

図12-6-12　TYPE2ロング・フォーマット・ページ・ディスクリプタ

63–48										
L/U	LIMIT									
RAL	WAL	SG	S	G	CI	L	M	U	WP	DT
PAGE ADDRESS (PA31-PA16)										
PAGE ADDRESS (PA15-PA8) (15–8)	UNUSED (–0)									

部分をページ・ディスクリプタと呼び，２ロング・ワードのショート型式と４ロング・ワードのロング型式とがあり，更にロング型式には，TYPE１とTYPE２がある(**図12-6-10，11，12参照**)。

この内には，ＡＴＣ内になかったフィールドやフラグがあるし，又，ＡＴＣ内に相当するビットもある。まず相当するものは，

ＳＧ：ＴＡＧ内のＳＧ（シェアード・グローバル）
Ｇ　：エントリのＧ（ゲート）
ＣＩ：エントリのＣ（キャッシュ・インヒビット）
Ｌ　：エントリのＬ（ロック）
Ｍ　：エントリのＭ(モディファイド)(書込みあり)
ＷＰ：エントリのＷ（ライト・プロテクト）

新たなものとして，

ＲＡＬ：リード・アクセスレベル・リミット

図12-6-13　ステータス・フラグU.Mのアクセスによる変化

Action by MC68851	Previous U	Previous M	R/$\overline{W}$	New U′	New M′
RMW Cycle to Set U (M Not Changed)	0	0	R	1	X
Write to Set U and M	0	0	$\overline{W}$	1	1
RMW to Set U	0	1	R	1	1
RMW to Set U	0	1	$\overline{W}$	1	1
No Write	1	0	R	1	0
Write to Set M (U Written Set)	1	0	$\overline{W}$	1	1
No Write	1	1	R	1	1
No Write	1	1	$\overline{W}$	1	1

図12-6-14　ディスクリプタ・タイプ・デタミネーション

DT FIELD	TABLE SEARCH STATE: TI FIELDS NOT EXHAUSTED	TI FIELDS EXHAUSTED	INDIRECT DESCRIPTOR SEEN
INVALID ($0)	INVALID	INVALID	INVALID
PAGE DESCRIPTOR ($1)	PAGE DESCRIPTOR TYPE-2	PAGE DESCRIPTOR TYPE-1	PAGE DESCRIPTOR TYPE-1
SHORT ($2)	TABLE DESCRIPTOR	INDIRECT	ILLEGAL
LONG ($3)	TABLE DESCRIPTOR	INDIRECT	ILLEGAL

WAL：ライト・アクセスレベル・リミット

この2個所の各々3ビットのフィールドは，モジュール・コールの許される最高位レベルを示している。

S：スーパーバイザ　スーパーバイザ・モードのアクセス時でのみ使用される事を示している。

U：アップデイト　このエントリが実際に使用された事を示す。

DT：ディスクリプタ・タイプ　次のレベルのディスクリプタのタイプを示す場合，このエントリの（以下も含めて）の無効を示す場合，このページ・ディスクリプタ自身がある場所を示す（間接指定）場合がある。これにより，ロングとショート，タイプ1と2を切り替える（図12-6-14参照）。

12-6-10　TYPE 1と2との差

この差は図12-6-15を見て戴くとわかるが，L/UのビットとLIMITのフィールドの有無である。これらは，ページの数，又大きさに制限を設けている。

このテーブル・ディスクリプタは，原則的には1ページに1個必要となる。もちろん，ある大きさを1個のディスクリプタで済ませる方法もある。これは，ATCは，必ず1ページ，1ATC対応となっているのに等しい。つまり，1ページ分のメモリには，1ATCが対応し，1ATCには原則1ページ・ディスクリプタが対応して，ページごとの保護とアドレスの変換を行なう。

従って，システムは物理メモリをページの大きさで分割した分の数だけは，このページ・ディスクリプタが作れるテーブル・エリアを確保しておく必要がある。

12-6-11　ページ・ディスクリプタへのアクセス

このページ・ディスクリプタ，つまり個々の変換用のデータは，PMMUがバス・マスタとなって自分で捜しに行く（この時をDMAモードと言う）。この手順は次の通りである。

1)．MPUのアクセスがPMMUにくるが，PMMU内のATCは全てHITしなかった。

2)．PMMUはMPUにバス・エラーとHALTを同時に与え，ASのネゲイトと同時にバス・エラーはネゲイトしてHALTは与えたままにする。これによりMPUは，リトライの待ちに置かれる。

3)．PMMUは，物理メモリ側のバスのアービトレイションをチェックして，他に使用者がいればその使用終了を待つ。いなければ，又はバスが空けばバス・マスタとなってルート・ポインタに示される物理メモリの検索を始める。

4)．変換テーブルのアクセスを始める。始めの条件はATCがHITしなかった時のアドレス，FCコード，R/Wである。つまり，このMPUからのアドレス等に対応する物理メモリを捜しに行く。この捜索過程で，これらのアドレス等は常にINDEXとして用いられる。ここで言うINDEXは，データ・テーブルの先頭のアドレス（ベース・ポイント）を与えられた時に，そこから何番目であるかを決めるものである。

5)．2番目の条件は，このデータ・テーブルのツリー構造の最上位に相当するPMMU内のレジスタであり，これが物理メモリ中のテーブルの位置を示す。このレジスタは3個あり，FC3＝Hで用

いられるDMA用ルート・ポインタ（ＤＲＰ），スーパーバイザ・モードでのみ用いられるスーパーバイザ・ルート・ポインタ（ＳＲＰ），一般のプログラムで用いられるＣＰＵルート・ポインタ（ＣＲＰ）がある（ＳＲＰは，PMMU内のＳＲＥビットがHのときのみ使用可）。

ＦＣコードにより，これら３本のルート・ポインタの一つが選ばれると，PMMUはこのルート・ポインタの示す物理アドレスを先頭とするテーブル検索を開始する。

5-1)．ＦＣコード

そのテーブル検索で使われる第一番目のINDEXはＦＣコードである。これはＤＲＰの時は必ず，このレベルが必要で，ＣＲＰ，ＳＲＰについては，ＴＣレジスタ内のＦＣＬビットにより，このレベル使用の有無を決定できる。先のＡ，Ｂエンジニアの例の様にプログラムとデータのアドレスが重複する可能性があり物理メモリ側でこのプログラムとデータが分けられていない場合には，このＦＣコード・レベルは必要となる。

このレベルは，ＦＣ３も含めると，４ビットで16個の次の階層のベース・ポイントを指定する「テーブル・ディスクリプタ」が含まれる。

16個のテーブル・ディスクリプタがあっても，その内の幾つかは，まったく使われない。これはＳとＵ，ＰとＤの組合わせで，MMUの変換に関るのは実際には，１ルート・ポイントにDMA時で８個，ＣＰＵ時に４個しかない。それ以外のテーブル・ディスクリプタは存在しなければならないが，使われる事はない。この場合，各ディスクリプタのＤＴフィールドをINVALIDとしておき，以下の階層のない事，従って変換不能を宣言しておく。これは，各階層で共通である（**図12-6-17参照**）。

5-2)．ＦＣコードを使わない場合には，ルート・ポインタは，直接テーブルのベース・ポイントを指定する。

この場合，又はＦＣコードによって選ばれたベース・ポイントより，その内のテーブル・ディスクリプタをINDEXを行なって選択する。この時はこのINDEXを行なうのは，ＭＰＵが出力してきたアド

図12-6-15　ルート・ポインタ・レジスタ・フォーマット

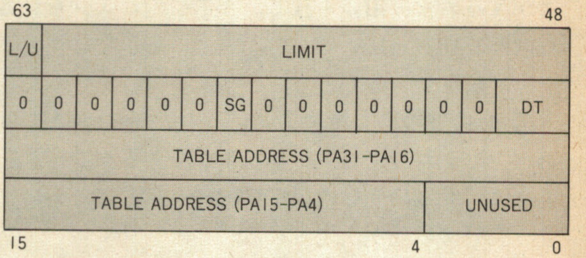
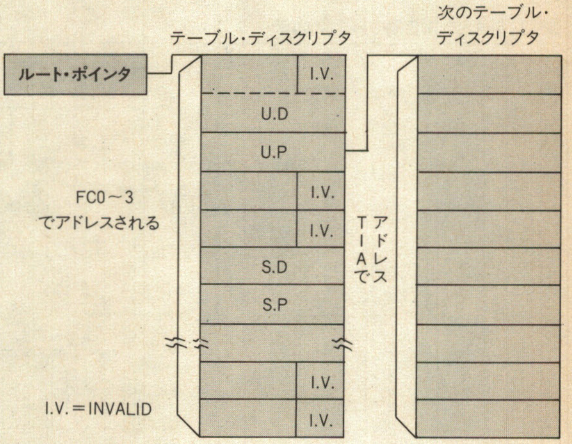


図12-6-16　FCコードとルート・ポインタ

FC3	FC2	SRE	Root Pointer Used
0	0	0	CRP
0	0	1	CRP
0	1	0	CRP
0	1	1	SRP
1	X	X	DRP

図12-6-17　テーブル検索手順



レスの内容であるが，このテーブル自体の階層構造の為に，この論理アドレスは幾つかに分割され得るしシステムが実際に必要とする論理アドレス範囲の限定も行なえる様になっている。

テーブル・ディスクリプタは従って，何段階かを通過してページ・ディスクリプタに達する。その階層レベルは一定ではないが各階層に割当てられているINDEXの為のビット数はＴＣ内で予め一定とされている。もし，ページ・ディスクリプタが，中間の階層で見つかると，それはそれ以下の階層は，一つの大きな連続したメモリ・ブロックである事を意

味し，指定され得ない下位のアドレスは論理アドレスが，そのまま用いられて，ATC内に入れられる(図12-6-18参照)。

5-3)．アドレス分割の方法

図12-6-19の様な論理アドレスの分割をPMMUは論理アドレスに対して行なう事ができる。

まず，左端のFC内はTC内のFCLビットにより，有効／無効が決定される。

次にISフィールド〔イニシャル・シフト〕(TC内)により，論理アドレス空間の大きさを決める事ができる。例えば，論理アドレス空間を68010と同じ16Mバイトとしたければ，IS＝8とすれば，A24～A31は使われなくなる。これにより，テーブルのサイズの最適化が計れる。又，これはモジュール・サポートで使用される上位3ビットとは無関係に単にATCへ入力されるアドレスの比較範囲を決定する為に用いられる。

次に右端のPS（ページ・サイズ）である。これは先に示したが，論理アドレスを物理アドレスとして，そのまま出力する範囲を指定する。

論理アドレス32ビットの内，このISとPSでせばめられた残りを（残りが0では不可）最大四つまでのINDEXに分割できる。この四つのフィールドは各々最大15ビットまでの値をとる事ができる。

この様子を**図12-6-20**に示すと，まず，ISにより全論理アドレス空間を16Mバイトとする。そしてTIAを3ビットとすると，これは16Mバイトの空間を8個の2Mバイトの空間に分割した事になる。そして，その内の一つの2Mバイトの空間はTIBを4ビットとすると，16個に分割された事になる。この時，その一つの大きさは128Kバイトとなる。このある一つの128Kバイトの空間をTICを2ビット

図12-6-18　3階層の場合のテーブル・ツリー

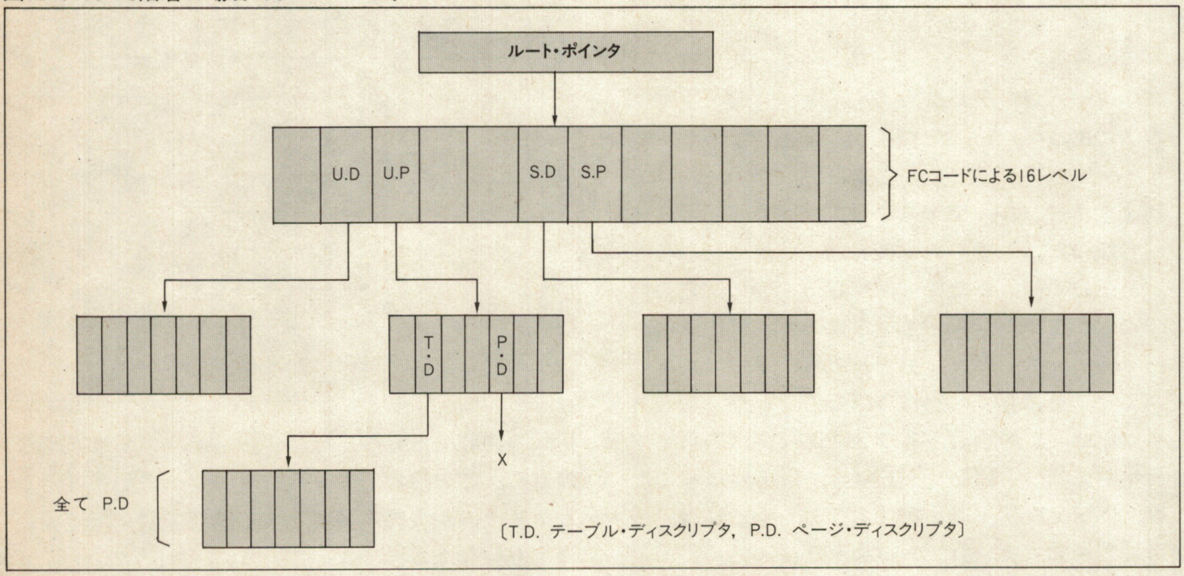
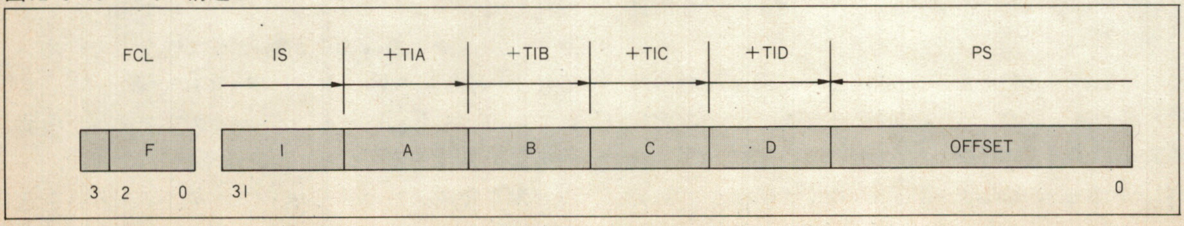


図12-6-19　ツリー構造とTCレジスタ



とすると4個の32Kバイトに分割した事になり，PSが32Kバイトならテーブル・ツリーの構成は，これで終わりとなる。この例の各段階ごとに分割された1ブロックごとに,テーブル・ディスクリプタか,ページ・ディスクリプタか，どちらか1個が対応していく。つまり，TIBの階層で，一つのテーブル・ディスクリプタがINVALIDを示せば，それはその128Kバイトに割当てられている物理メモリは全くない事を示している。同様にTIAのレベルでの一つのディスクリプタのINVALID宣言は，2Mバイトが無効である事を示す。逆にTIBの階層のページ・ディスクリプタは，TIC階層はこの部分にはない事を示し,128Kバイトの連続した同じ保護内容の物理メモリが割当てられていることを示している。

こういった事がなく順当な場合には，最下位の1ページ・ディスクリプタが，1ページ分の空間の変換すべき物理アドレスと保護内容を与える。又，この内容を，現有の物理メモリ分だけ予め作っておきその最下層から，更に間接指示で指定する事も可能である。いずれの場合のPMMUは検索を行なって，ページ・ディスクリプタを見つけ，そのデータ（物理アドレスとステータス・フラグ）をATC内に取り入れる。この時，最も古いものが捨てられて置き換えられていく。

6)．ATC内に論理アドレスに対応した新たなATCができると，PMMUはMPUに対するHALTを解除して，先のメモリ・サイクルを再開する。もし，この新たなATCの内容とアクセスの条件が合わないと，バス・エラーにて再開されたサイクルは終結させられる。

7)．ページ・ディスクリプタの検索中にINVALIDを発見した時，PMMUは，やはりバス・エラーを発して，このメモリ・サイクルを終わらせて，MPUによるテーブルの再構成を可能とする。

12-6-12 ルート・ポインタの役割

各タスクは自己の使っている論理アドレスと，物理アドレスを変換するテーブルを独自に必要とする。

図12-6-20 インデックス・アドレスとテーブル・ツリー

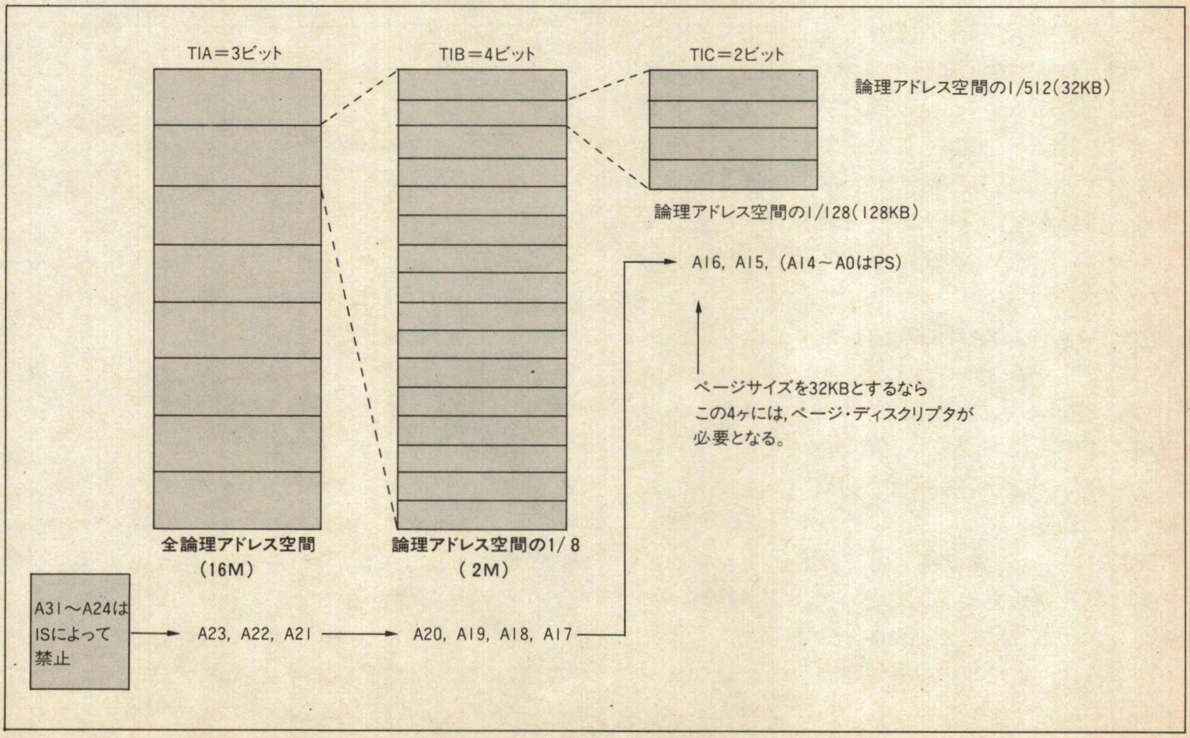
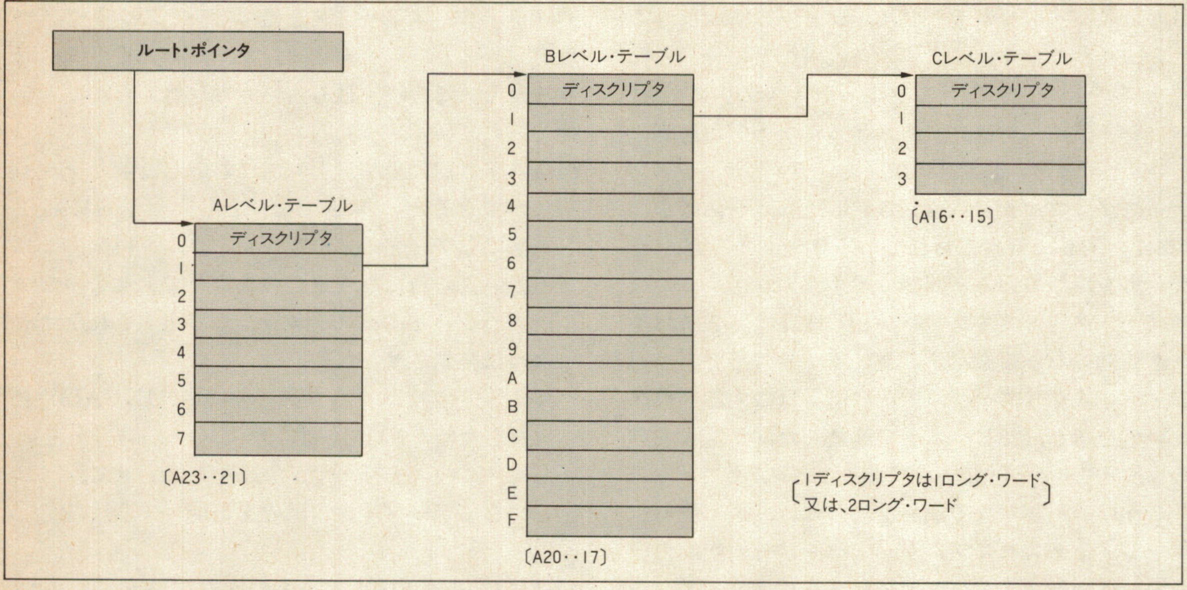


図12-6-21　テーブル・ツリー内のディスクリプタ



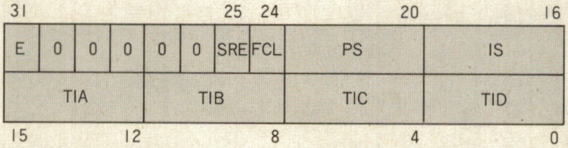
そこで，ＯＳのPMMUドライバは，新しいタスクごとに，この変換データ・テーブルを作り出し，その最上位のテーブルのアドレスを，ルート・ポインタ・レジスタに入れて，その時に何処のテーブルが使われるべきかを指示する。仮想マシンを構成する時には，物理メモリの量は不足するので，２次記憶に出力した部分のページ・ディスクリプタには，INVALIDマークを付けて誤用を防ぐ。又，この時，関連するＡＴＣもPFLUSHR命令等を使って同時に消去しなければならない。その後，その無効とされたメモリ部分は，他のタスク用に供され得る。

12-6-13　PMMUのＡＴＣ関連レジスタ

前述のツリー構造の各段の深さや，数を決めるのがＴＣレジスタである。この内の各ビットやフィールドは既に説明されているが，最上位のＥビットのみは言及されていない。これはＰＭＭＵの変換機能の開始ＳＷである。

PCSRというレジスタもあり，これの主な目的はＣＲＰに新しい値を書き込んだ時に，消されたタスク・エリアがあれば，これを表示する事である。又64エントリ中63個にロックが成された時の警告ビットも有する。

図12-6-22　トランスレーション・コントロール・レジスタ・フォーマット



E — ENABLE
SRE — SUPERVISOR ROOT POINTER ENABLE
FCL — FUNCTION CODE LOOKUP ENABLE
PS — PAGE SIZE
IS — INITIAL SHIFT
TIA, TIB, TIC, TID — TABLE INDICES

後の変換関連は，PTEST命令でのみ用いられるＰＳＲであり，これはオペランドで示される変換に対しての各種情報を教えてくれる。

PMMUは，その制御レジスタに関しては以上の様に簡単である。

12-6-14　68851のコ・プロセッサ命令

PMOVE：PMMUのレジスタのＲ／Ｗ
PVALID：モジュール・サポートの階層を表すアクセス・レベルの現在値に対しアドレス値が正当かをチェックする。正当でなければ例外処理に移行する。
PTESTR：論理アドレスをリードする時に対応するテーブル検索を行なって，そのステータスとアク

セス・レベル値をPMMUのステータス・レジスタに表示する。

PTESTW：論理アドレスをライトする時に対応するテーブル検索を行なって，そのステータスとアクセス・レベル値をPMMUのステータス・レジスタに表示する。

PLOADR：指示した論理アドレスを変換するためのテーブル検索を行ない，変換データを変換用キャッシュ・レジスタにセットする。変換用テーブルの履歴は使用実績となる（リード・サイクル）。

PLOADW：PLOADRと同様にキャッシュにデータを入れる。変換用履歴には，このページの内容が変更された実績となる（ライト・サイクル）。

PFLUSH：論理アドレス・FCコードや，FCコードとアドレスの組合わせにより，指示されるキャッシュ・レジスタの内容を消去する。

PFLUSHA：変換用キャッシュ・レジスタの内容を全て消去する。

PFLUSHR：ルート・ポインタに示されるルート・ポインタ・テーブルとこれに関連するキャッシュ・レジスタの内容を消去する。

PFLUSHS：論理アドレス，FCコード，またはその組合わせによる変換キャッシュ・レジスタの内容をグローバルなエントリの場合も含め消去する。

PSAVE：仮想マシンをサポートするため，PMMUの内容を全て外部のメモリ中に保存する。

PRESTORE：PSAVEによって保存したデータをPMMUに戻す。

PBcc：PMMUによる条件判断でプログラム・フローを切替える。

PDBcc：MPUのDBcc命令に相当，ただし，条件判断のみPMMUが行なう。

PScc：PMMUの状況により，オペランドをセットする。

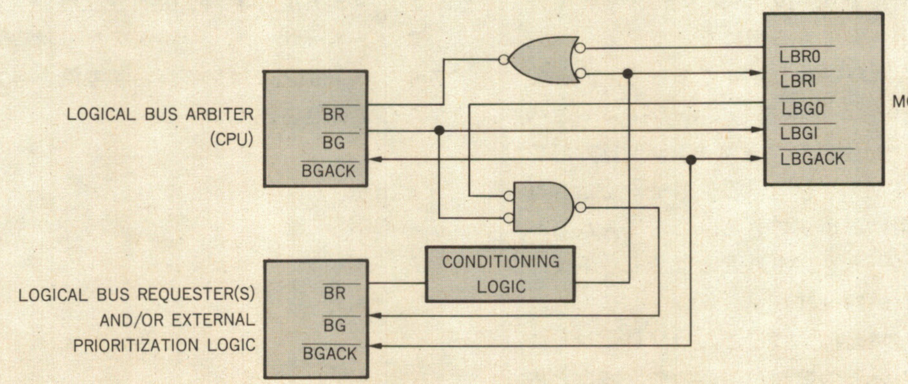
TRAPcc：PMMUの状況により，例外処理に移行する。

12-6-15　PMMUの周辺ディバイス

PMMUと関連の深いディバイスは，他のコ・プロセッサ（FPCP等）とDMACそれに，キャッシュ・メモリであろう。特にFPCPは多用されるが，その接続は論理空間に置いても物理空間に置いても問題はない。

DMACは，現在の所68020用のDMACとしての専用ディバイスはないのだが，論理アドレス側に置くと，PMMUからの各種保護が受けられる。物理アドレス側に置くと，全てを物理アドレスで処理できて高速となるが保護には欠ける。いずれにしてもよいのだが，いずれの場合も，バスのアービトレイションは，バス・マスタがMPU，PMMU，DMACと三つの組合わせとなるので，注意を要する。この点はPMMUのユーザーズ・マニュアル，付録Bを参照さ

図12-6-23　論理バス側での3者のアービトレイション



れるとよい(図12-6-23)。

同様の問題が外付のキャッシュ・メモリにも存在する。物理側に置けば，アクセス・タイムは落ちるが，アドレスの多重化の問題はない。論理側に置くと，68020の3サイクルでの応答も可能だが，タスクが変わるごとに,フラッシュされなければならない。このあたりの細かいタイミングの議論と用例はやはり，ユーザーズ・マニュアルの付録Bに詳しい。

12-6-16　モジュール・コールのサポート

CALLM命令に付いては，命令の章で一項を設けて解説したが，実際に階層を設けてのシステムの保護は，このPMMUなしにはできない。

PMMUは，内部に現在のアクセス・レベルを有し，各ページ・ディスクリプタにも各々，許されるアクセス・レベルを記述させ，スタックの変更のチェック，アクセス・コントロール・レジスタによる制御を行なう事によって，ソフトウェア上の許されない移行に対して防御を行なっている。

但し，このアクセス・レベルを現在使用している例は，余り見られない。ソフトウェア上の複雑さとこのPMMUの普及度の低さの為と思われる。

この件は，PMMUのユーザーズ・マニュアル第10章を参照してから採否を考えられると良い。

12-6-17　BKPT命令

このBKPT命令についても命令の章で解説した。PMMU内には,代替命令を入れておくレジスタが八つ，又，その各々に制御用のレジスタが付いて，BKPT命令をサポートできる様になっている。

PMMU自体が，コ・プロセッサであって，内部で自分のアクセスのデコードをしている様に，このBKPT命令専用空間へのアクセスもPMMU内部で行なわれるので,PMMUを用いるシステムでは容易にBKPT命令を用いる事ができる。ここに入れるべきデータ，パラメータをPMOVE命令でセットするのみで良い(図12-6-24～26)。

以上がPMMUの解説である。このMMUに対しては複雑で判りづらいとの批判が多いので，概念と内容の対比に心がけたつもりであり，詳細な点は割愛

図12-6-24　モジュール・コールをサポートするレジスタ

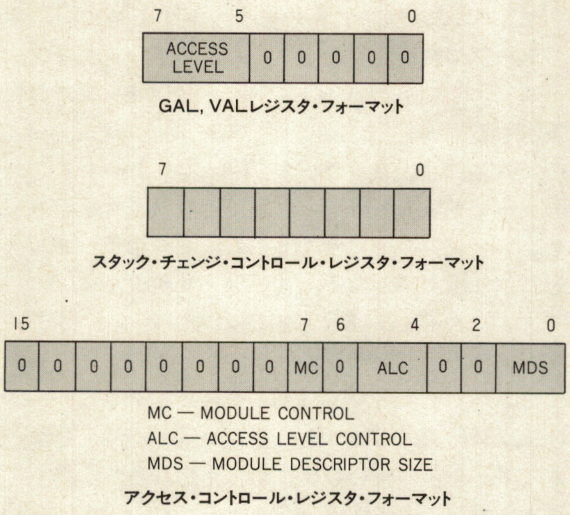


図12-6-25　ブレーク・ポイント命令用代替命令レジスタ

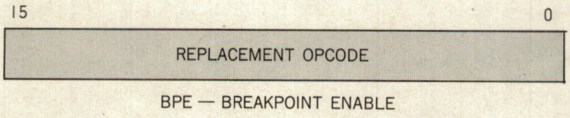
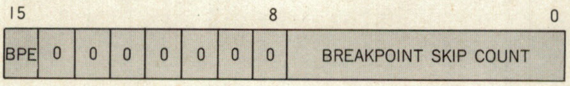


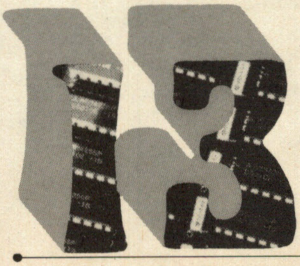
図12-6-26　ブレーク・ポイント命令コントロール・レジスタ



した。ここで説明した概念がわかっていればユーザーズ・マニュアルの単調な説明も理解できるであろう。また，ユーザーズ・マニュアルの付録A，Bはアプリケーションに対して良い参考例を与えてくれるから必ず参照されたい。

第13章

MC68030の機能解説



13-1　MC68030の概要

68030は68020をベースに作られたフル32ビット・アーキテクチャのMPUである。その大きな違いは以下の2点であるが，この為に，各種の細かい変更がなされており，総合的には，かなり上位のプロセッサとして仕上がっている。

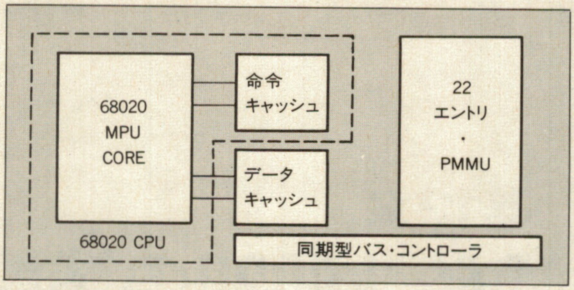
- PMMUのサブ・セットを内蔵した
- データ用のキャッシュ・メモリを内蔵した

この2点においてもまずMMUを内蔵しているのであるから，当然仮想マシンを構成する事になる。従って，UNIX等のOSが使われるEWS等の規模のマシンに用いられると考えられ，小型の単能機には余り用いられる事はないであろう。

そこで，以下のデータ・キャッシュを含む改良点が重要になってくる。つまり，この68030というMPUのコンセプトは完全に32ビットのメモリ幅を持っている高速のメモリが使えるという前提で考えられている。この事を前提とした使用法や，周辺回路の設計を行なわないと，せっかくのこのMPUの個性を生かせないことになる。

この他に，68030では，外部からデバックの為にチップを監視し，その実行状況を知る事ができる様に，2本の出力線が追加されている。これは，$\overline{\text{REFILL}}$と$\overline{\text{STATUS}}$である。

図13-1-1　68030の内部構成



この2本の信号は，両方共にMPUの内部の命令実行の為マイクロ・シーケンサの実行状況を示していて，これにより各命令の実行状況を知る事ができる。$\overline{\text{REFILL}}$は，内部の命令パイプの内容が捨てられ，新たな内容のパイプへの導入が始まった事を示しており，これは，プログラムがその連続的な実行アドレスを変えた事を意味し，$\overline{\text{STATUS}}$信号は，そのクロック数によって**表13-1-1**の様に種々の意味を持つが，最も一般的な意味として命令の境界，又は始まりを示すものとして扱える。これらの信号を使う事によって，より容易にMPUの実行状況，エラー状況を知る事ができる。

13-2　PMMU68851との違い

68020用であり，外付けのPMMUである68851を使用した場合には，各メモリ・アクセスに対して必ず

表13-1-1　STATUS信号によるMPUの動作状況

Asserted for	Indicates
1 Clock	Sequencer at instruction boundary — will begin execution of next instruction
2 Clocks	Sequencer at instruction boundary but will not begin thenext instruction immediately due to: • pending trace exception OR • pending interrupt exception
3 Clocks	MMU address translation cache miss — processor to begin table search OR Exception processing to begin for: • reset OR • bus error OR • address error OR • spurious interrupt OR • autovectored interrupt OR • F-line instruction (no coprocessor responded)
Continuously	Processor halted due to double bus fault

1WAITが入り，最短メモリ・サイクルは4クロック・サイクルとなっていた。この点は68030MPUでは改良され，又，オン・チップの効果は大きく68020の本来のメモリ・サイクルである3クロック・サイクルで，非同期アクセスを行なう事ができる。つまり，通常のバス・アクセスは，68020と同じになっている。

68851を使う時に，複雑であったアドレス信号，LAnとPAnとの関係はなくなり，68020と同じ単なるアドレス信号となった。

更に68851を用いる場合で，外付論理回路として最も複雑になるバス・アービトレイションの複数化による問題がなくなり，これも通常の外部バス・マスタとMPU間のみのアービトレイションとなった。これはPMMUとMPUの間でアービトレイションはやはりチップ内部で解決されているからである。

そこで，68030はPMMU内蔵であっても，実にすっきりとした入出力ピン構成となっていて，PMMUが内蔵されている為に68020と異なるピンは内蔵のPMMUを禁止してしまう為，「MMUDIS」ピンのみである。このピンをLにすると内蔵のPMMUは使われず，MPU本体のアドレスが直にアドレス・バスに出力される。

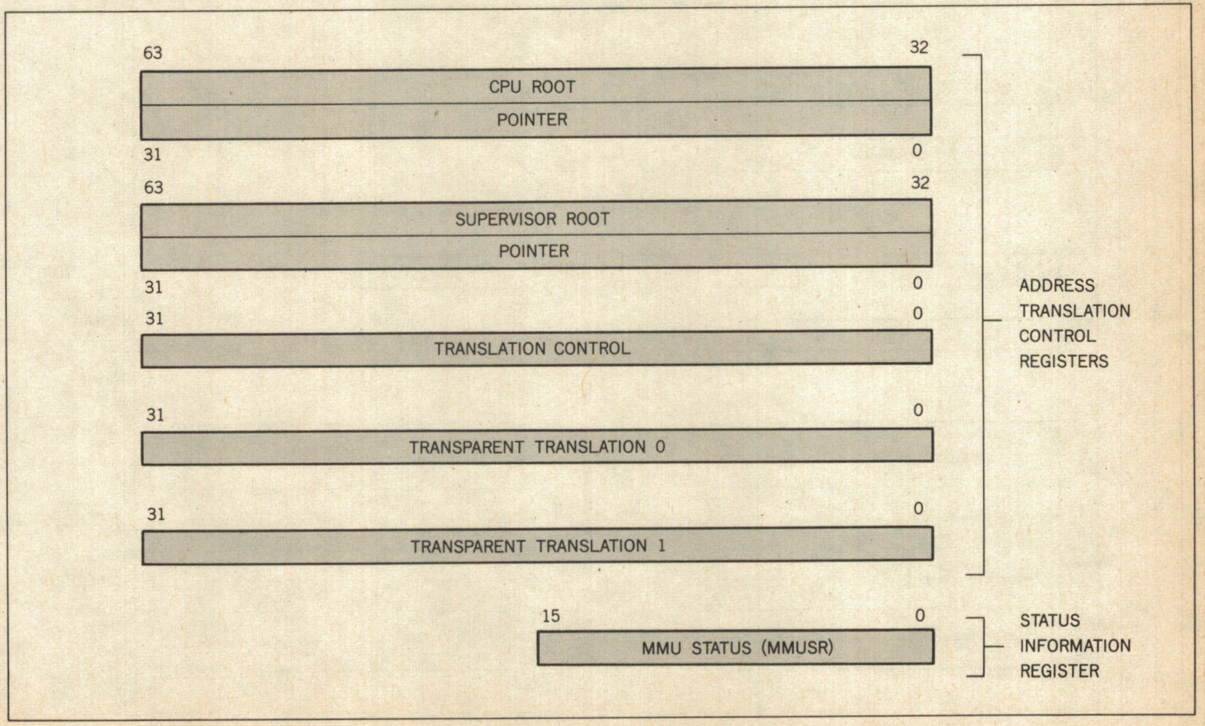
もう一つ解決されているものとして，RESET時のPMMUのステータス読込みがある。PMMUは内蔵されているので，MPU本体と常時32ビットのデータ・バスとアドレス・バスで結ばれているので，RESET解除時に読込む必要のあるものはなくなってしまったからである。

13-3　MMUの性能

前項では主として外から見た68851と内蔵PMMUとの違いについて述べたが，ソフトウェア的にも，変換能力についても，差異は存在する。

MMUがMPUからの論理アドレスをメモリ用の物理アドレスに変換する方法に関しては，まったく同じである。しかし，そこで使われるアドレス変換テーブル（物理メモリ上に置かれる）を示すルート・ポインタの数が，まず減らされている。これは68851

図13-3-1　68030内のMMU関連レジスタ



はファンクション・コードを4ビット・サポートしていたので，この4ビット目はDMA用に割当てられていた。しかし68030の内蔵PMMUは，外部のDMACが出力するアドレスを内部に取込んで変換する事は不可能である。これは，改めて述べるまでもないが，内蔵PMMUのアドレス入力は全てMPUのアドレス出力に接続されているからである。

以下に68851PMMUと68030内蔵MMUとの差を列記する。

－1：DMAルート・ポインタの削除（前述）

－2：BKPT命令サポート回路の削除

－3：エントリ数の減少（64→22）

－4：MMUキャッシュ・ステータスの削除

－5：アクセス・レベル・コントロールの削除

－6：MMUキャッシュの固定機能の削除

－7：タスク・エリア，フィールドの削除

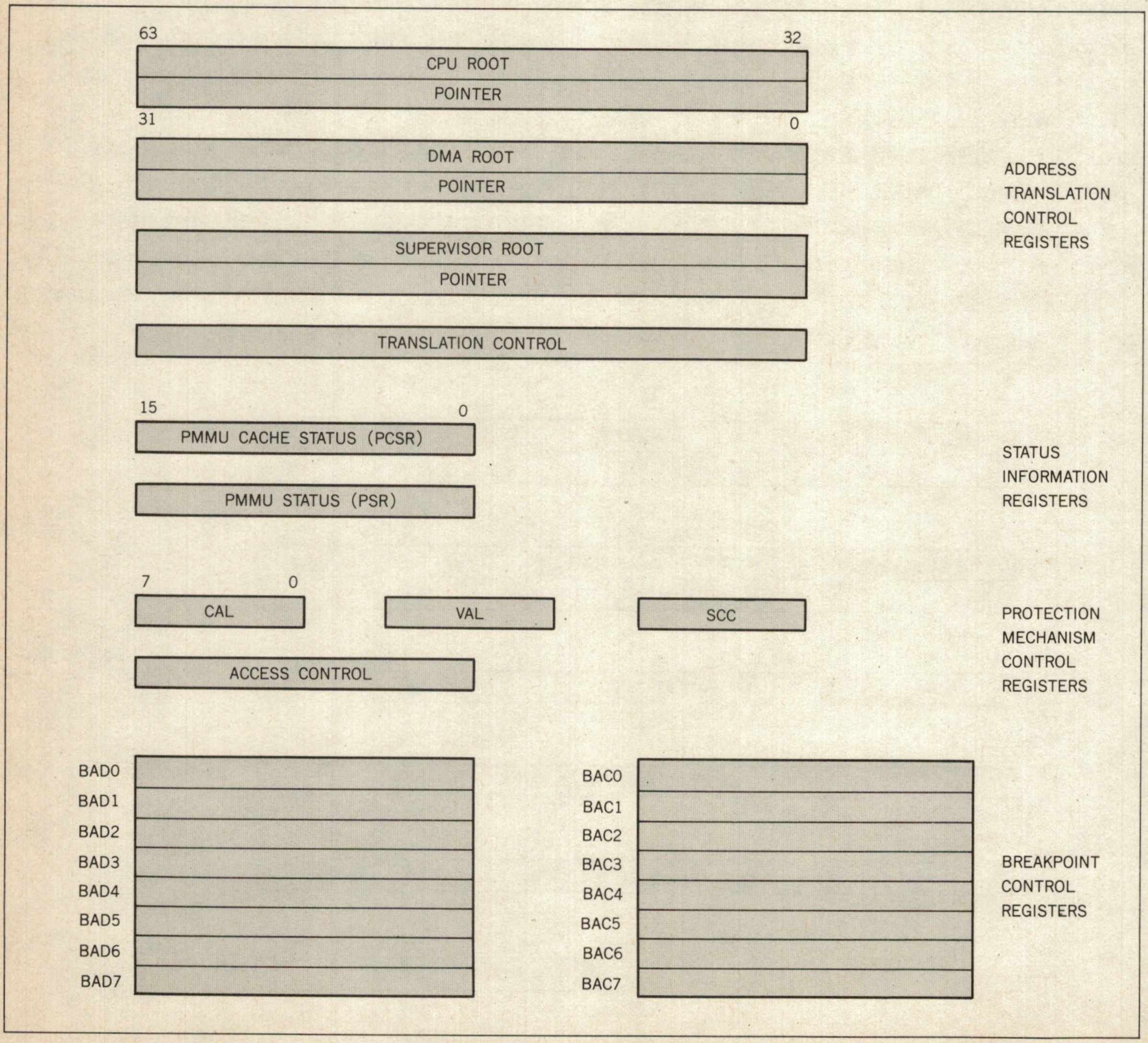
－8：SG（シェアード・グローバル）の削除

－9：ルート・ポインタ・テーブルの削除

－10：コ・プロセッサとしての機能の削除

（PSAVE，PRESTORE，PBcc，PDBcc，

図13-3-2　68851PMMUの内蔵レジスタ



PScc，PTRAPcc)
＋１：トランス・ペアレント・レジスタの追加
＋２：ＣＩ（キャッシュ・インヒビット）の追加
これらの変更は主として，このＭＰＵがＥＷＳ等の高速のシステムで，少ないユーザー数をサポートする様に考えられているからである。もし，大きなシステムを構成するなら，機能の高いMMUである68851を使ってシステムを構成すればよい。

－１）は先に述べた様に，ＤＭＡを接続する事が不可能であるからである。

－２）は，BKPT命令のサポートは必修のものではないので，外付けとしチップ上の集積度を下げている。

－３）は，やはり集積度の低下の為であるが，また，同時にマルチ・タスクの他のタスクのエントリを生かしておく必要性の低下により，数を減らしても良いとの判断の為でもある。

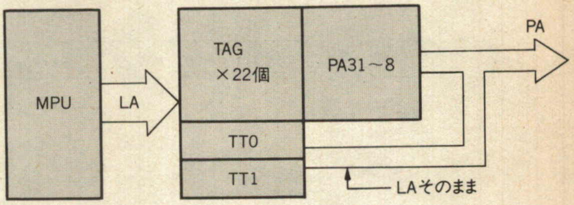
－４）～－９）は，いずれも同じ発想である。ＣＲＰに連結したルート・ポインタ・テーブルはマルチ・タスクをサポートする為の機構であったが，マルチ・タスクのスイッチングの度に全部の内蔵ＡＴＣを消してしまっても不都合がなければ（もちろんMMUのＤＭＡの為に若干遅くはなるが）このテーブルは不要で，そのテーブルに持たれていた，タスク・エリアも不要となる。そこで，共通タスクの為のＳＧも不要となる。

また，22個のＡＴＣしかないのであるから，あるエントリを固定しておく余裕はない。高度の保護メカニズムをサポートする様なモジュール・コールの必要性も低下するので，アクセス・レベル・コントロールも削除し，従ってCALLM，RTM，PVALIDの命令も削除された。

－10）は，この内蔵MMUはコ・プロセッサではないので，コ・プロセッサの通信プリミティブによる命令とその機能は削除されている。そしてＣＰ・ＩＤ＝０（68881用にリザーブ）は，F-LINEの例外処理を引き起こす事になる。この68030の外側にPMMUを付ける事は従って不可能である。(MMUDIS＝Ｌとしても)。

これに対して，MMUは比較的小型のシステムで

図13-3-3　68030MMUのアドレス変換



都合の良い性質が追加された。

まず，RESET時にMMUは完全に禁止され通常のReset-Startを許す。そして，内蔵のデータや命令キャッシュと同時に，MMUのＡＴＣはアクセスされ，メモリ・サイクルの減少に役立っている。又，追加されたバースト・モードでも正しく動作する。

新しい機能としてトランス・ペアレント・レジスタが２本追加され，１本に付き64Kバイト，又はその2^nの単位でアドレス変換を行わない（論理アドレス＝物理アドレス）メモリ空間を作る事ができる。この場合,，リード・オンリや，ライト・オンリにする事も可能となり，アドレス変換テーブル自体のメンテンスや，ＲＯＭ化されているＢＩＯＳへのアクセス等に有効である。このトランスペアレント・レジスタ内にも，キャッシュを禁止するＣＩビットがあり，これを用いてのアクセスにキャッシュを使わない様にする事ができる。

この機能により，この内蔵MMUはより簡易に使えるものとなっている。アドレス変換テーブルの作り方などの詳細な点は，PMMUと同じであるので割愛する。

13-4　68030のキャッシュ・メモリ

68030のキャッシュ・メモリは，68020では命令用のみであったものにデータ用が追加されるという改良があった。これに伴ってキャッシュ自体の内容に変更が加えられただけでなく，外部に更に拡大されるであろうローカル・キャッシュの為の制御信号やキャッシュの内容のアップデイトの為のバースト・モードの追加，このバースト・モードで用いられる高速のバス・アクセス方式の追加と，このキャッシュと関連する変更は数多く行われている。これらの

図13-4-1　68030キャッシュ・コントロール・レジスタ

31–14	13					8	7		5	4				0
000000000000000000	WA	DBE	CD	CED	FD	ED	0	0	0	IBE	CI	CEI	FI	EI

WA＝Write Allocate
DBE＝Data Burst Enable
CD＝Clear Data Cache
CED＝Clear Entry in Data Cache
FD＝Freeze Data Cache
ED＝Enable Data Cache
IBE＝Instruction Burst Enable
CI＝Clear Instruction Cache
CEI＝Clear Entry in Instruction Cache
FI＝Freeze Instruction Cache
EI＝Enable Instruction Cache

改良部分は全体としてより高速の実行を可能とする事を目的としており，この範囲では，MMUを除いても68020の上位と言える。

13-4-1　MPU内蔵のデータ・キャッシュ

ＭＰＵの内部にデータ・キャッシュを入れる場合には幾つかの難点がある。それは単に固定した内容が読まれるのみである命令コードをＭＰＵ内のキャッシュに入れておく場合と異なり，オペランドで参照されるデータは時々刻々と変化してゆくものであり，その変化を発生させるのはＭＰＵのみではないという点にある。外部との通信や，状況の読み出しに用いているＩ/Ｏディバイスからの読み取った内容をキャッシュに入れてしまうと，これらＩ/Ｏからの新たなデータが読めなくなり，キャッシュ内の固定されたデータのみを使う事になって不都合であるし，「ＴＡＳ」命令で用いられる様なメモリの内容をキャッシュ入れてしまったら，せっかくの「ＴＡＳ」命令が正しく働かない。

以上の例で判る様に，データ・キャッシュへの内容のアップデイトは，ダイナミックに禁止されたりイネーブルされたりする必要があり，その条件は外部的にも内部的にも存在する。

これと同様に，ＭＰＵの外部への書込みサイクルの問題も発生する。オペランドに用いられるデータであれば，当然そこに対し書込みが発生する。この書込みは，必ず外部メモリに行わなければならないが，同時に内部キャッシュもその対象となり得る。

ごく短いサブ・ルーチン・コールのリターン・アドレスや，プログラム内で連続的に使用する変数を入れておく所であればキャッシュに入れておくことに大きな意味があるが，他のプログラムに渡す為のデータ・テーブルの作成時には，書込みを行なった所を読出す事はまずあり得ないので，この書込みを全てキャッシュにも行なうと，無駄な事をしている事になる。この為，これを切換える制御ビットも必要となる(ＷＡ：ライト・アロケイト)。

13-4-2　キャッシュ制御の為の外部信号

68020では，キャッシュ制御の為の信号は，キャッシュの動作を禁止する$\overline{CDIS}$信号のみであり，この信号はキャッシュ全体の動作を停止させる為のもので各メモリ・アクセスに向けたものではなかった。68030には，各メモリ・サイクルの為に新たに$\overline{CIIN}$信号（キャッシュ・インヒビット・イン）が設けられ各アクノレッジ信号以前にこの信号をアサートする事によって，そのメモリ・サイクルへのアクセス(読出し，書込み）がキャッシュに取込まれなくなる。従ってＩ/Ｏに割当てたアドレス空間や，他のＭＰＵ等と共用しているメモリ部分，デュアル・ポート・メモリ等への選択信号は，ＯＲをとられてこの入力信号をドライブする必要がある。つまり外的要因で変化する内容を持つメモリ空間に対しては，$\overline{CIIN}$をドライブして，その内容がキャッシュに入らない様にする必要がある。更に，キャッシュは常にロング・ワードのデータ／命令を扱うので，読まれるポートの幅が32ビットをサポートできない場合（複数回に分けても，32ビットを与えられない場合。１アクセスではない。複数回のアクセスはバス・コントローラによって行なわれる）にも$\overline{CIIN}$信号を与える必要がある。この機能を補佐するのは$\overline{CIOUT}$信号である。この信号はMMU内のＡＴＣに含まれるＣＩビット（キャッシュ・インヒビット）が，MMUを用

いた外部アクセス時に出力されてくるものであり，この信号がアサートされると，そのメモリ空間（ページ空間）に対しては，内部的にキャッシュの動作は禁止される。従って適切なアドレス変換テーブルには，必ずこのＣＩビットの設定を行ない，先の要因に基づく場合，又はＣＰＵスペース，$\overline{\mathrm{RMC}}$サイクル等の内部理由に基づく場合等があるが，これらに用いるメモリ空間に対応する変換テーブルの相当ビットにＣＩビットを立てて，キャッシュの動作を禁止できる(外部キャッシュも同様)。

ここで注意すべき点は，例えば，あるＩ/Ｏの内容を一度キャッシュに入れてしまうと，その後テーブルを書換えても，そのキャッシュのアドレスＴＡＧが無効とならない限り，外部へのアクセスが起こらないので，ＭＭＵは使われず，従って，ＣＩビットも参照されない。

これはコンテクスト・スイッチを行なった際によく発生するミスであるが，特にデータのキャッシュとＭＰＵを内蔵した68030で多く発生する問題であると思われる。

従ってＭＰＵのルート(変換テーブルのポインタ)を変更した時には，必ずキャッシュの内容のフラッシュ（全無効）を行なう必要がある。

13-4-3　キャッシュの構造

68030のキャッシュ・メモリは，第12章で述べた68020のキャッシュ・メモリと構造上にも若干，差がある。もちろん，データ・キャッシュと命令キャッシュの違いは別としてである。その大きさは命令やデータを入れておく部分（エントリの大きさ）は256バイトと変わらないが，各エントリを選択するタグの数と，一つのタグに含まれるエントリの数が変えられている。68020では，１エントリが32ビットで(これは同じ)，１タグに１エントリであった。このタグは64個あってＦＣ２，Ａ31～Ａ08がこのタグの内に納められ，Ａ07～Ａ02で64個のタグの内１個を指定していた(図13-4-2)。

68030のキャッシュは，このタグの数が16個に減らされた。Ａ07～Ａ04がこの16個のタグの内の一つを指定して，そしてこの一つのタグで四つのエントリを持ち，各エントリごとにＶフラグがあって，そこのキャッシュ・エントリの内容が有効か否かを示している。この四つのエントリはＡ03とＡ02で指定される。これで何が違うのかというと，各エントリの

図13-4-2　68020キャッシュの構成

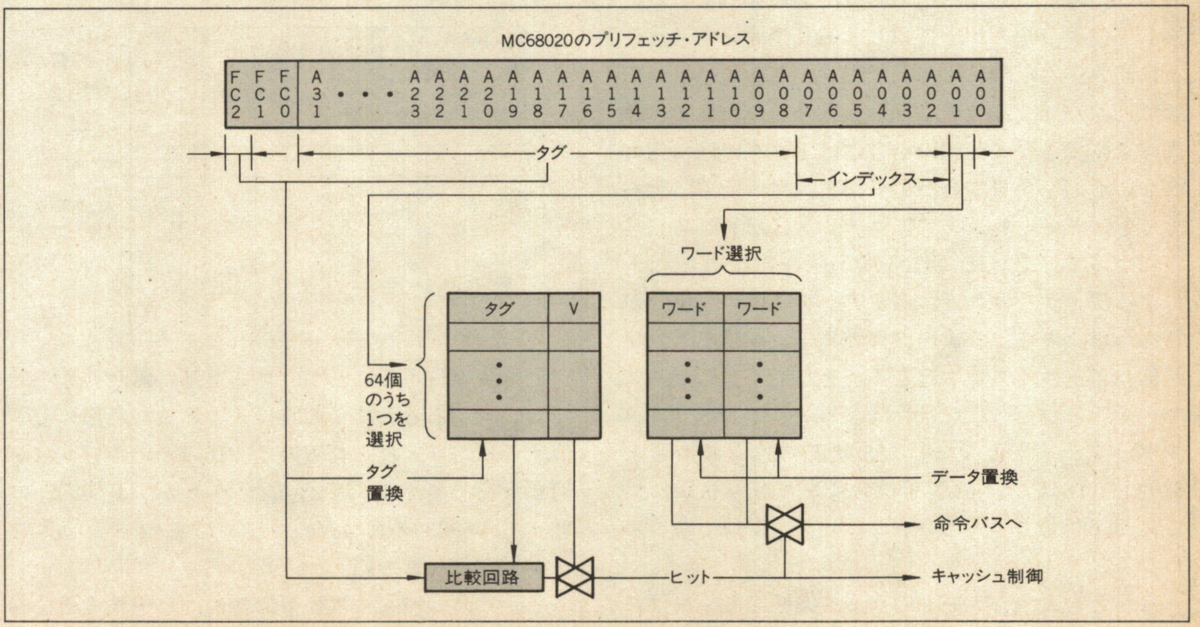
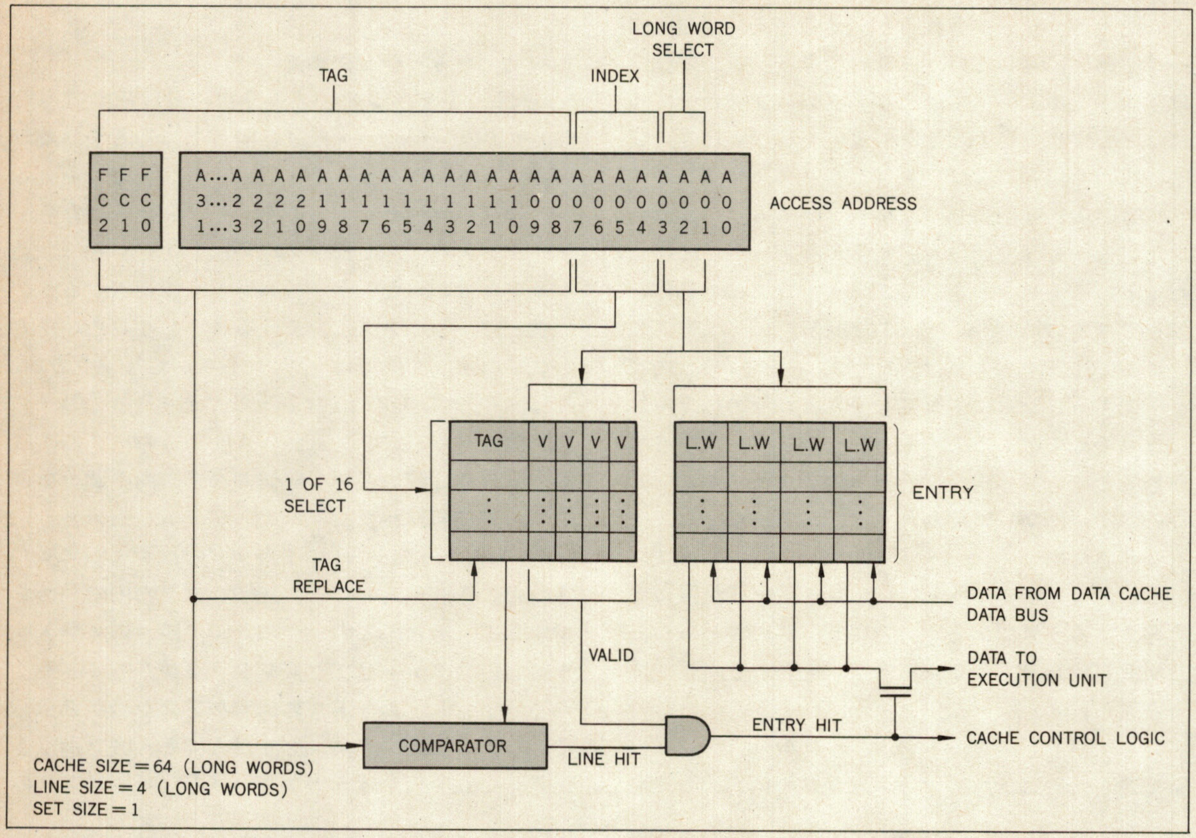


図13-4-3　68030のキャッシュの構成



有効性の差とアップデイトの方法論が違ってくる。もし上位アドレスが全て同じメモリ空間をキャッシュに読み込んだ時には，同じ256バイトが読込まれる訳であるが，68020のキャッシュは全て使われるべき内容を持てるが，68030の場合には１タグの内に使われないものを含む場合があり得る。命令でも，データでも，キャッシュのアップデイトはＭＰＵの読込みによって行なわれ，$\overline{\text{CIIN}}$やＭＰＵ内のＣＩビットによる禁止がなければ，あるアドレスへの読出しが行なわれ，キャッシュのタグの不一致が検出されて外部バス・サイクルが起きる。そして，もしバースト・モードがとられなければ，単一のエントリのみのアップデイトとなる。この状況のみを考えると68020では64タグ＝64個所の命令をキャッシュに持てたが，68030では同じ命令なら16個しか持てない事になる。

もっとも命令やデータがうまく連続していればタ

図13-4-4　キャッシュ内の構造

A3,A2		0,0		0,1		1,0		1,1
TAG	V	ENT0	V	ENT1	V	ENT2	V	ENT3
	1		0		0		0	
	1		1		0		0	

16個（TAG列の行数）

4ロング・ワード中1個のみ有効（1行目）

4ロング・ワード中2個有効（2行目）

グが一致（ＨＩＴ）して，エントリのみがバリッドでない時があるが，この時はこの連続した部分もエントリに取り入れられるので，16以上のエントリが有効となってくる。但し，最悪の場合には，68020に較べて，68030の命令キャッシュは１／４の有効エントリしか使われない。

しかし，一般的には命令は特に，データも連続的

な構造を持つものが多い。そこでキャッシュのタグ構造をこの様に変えても，ＨＩＴ率の低下は余りないと考えられている。それよりも，この新しい構造により，１タグの４エントリの内容を連続的に高速で更新するバースト・モードの採用とキャッシュ自体のメモリ・ビットの低減による高効率化の方が重視されている。

このタグの大きさを比較してみると（命令用），68020の64（タグ）×24（ビット）＝192バイト必要とするが，68030のキャッシュでは16（タグ）×24（ビット）＝48バイトとなり，192－48＝144バイトの節約になり，これが命令とデータ（データの方が正確にはタグは大きいが）で，290バイト以上となる。そこで，命令のエントリの大きさは，256バイトであるから，丁度，キャッシュのエントリ分以上のメモリ・ビットの節約となっている。つまり，もし68020と同じ構成のキャッシュを68030の命令とデータの双方に採用した場合に比べると，３分の２の大きさででき上がっている事になる。

13-5　バースト・モード

更に，この68030の１タグに４エントリというエントリ形式にはもう一つ重要な動作との関係がある。一つのタグがヒットしなかった時，キャッシュ・コントロール・レジスタ内のバースト・モードのイネーブル・ビットが立っていれば，68030はバースト・モードにてこの４エントリの連続した更新をメモリ側に対して起こそうとする。そして，もしメモリ側がこのバースト・モードに対応できれば，68030に新たに追加された同期型のバス・サイクル（最短２クロック・サイクル）にてＡ02，Ａ03の組合わせで作られる４ロング・ワードの内容をキャッシュに取り込んでしまう。これは丁度DRAMのニブル・モードでのデータ・アクセスと全く同じ動作を要求する事になる。例えば，＄00001004へのアクセスで，バースト・モードが発生すると，まずこの＄1004番地が読まれ，次に＄1008番地，＄100Ｃ番地と読まれて，最後に四つ目として＄1000番地に戻って読まれる。もちろん，四つ共にＶフラグは有効となり使用できるが，不要部分が入ってしまう事もあり得る。このモードでは，キャッシュのインデックスの範囲Ａ04で選ばれる以下のバイトは，１タグに全て取り込まれる事になる。

13-5-1　バースト・モードの制御信号

このバースト・モードを要求する場合，ＭＰＵは「CBREQ」信号をアサートして外部メモリに対しバースト・モードの要求である事を示す。そこで，もし，これに対応できれば，「CBACK」をアサートしてバースト・モードを受け付けたことを示す。そして，このバースト・モードでは，同期型のバス・サイクルが要求される。この同期型のサイクルはバスのメモリ幅32ビットの時のみに使えるデータ転送の方法で，ＭＰＵのクロックに外部で同期した信号をＭＰＵに与えて調歩同期をとるバス・サイクルを用いる。この場合でもWAITを入れる事は可能であるが，その最短の場合２クロックで１メモリ転送を終了できる。ここに関わるＭＰＵへの各入力信号（$\overline{\text{STERM}}$,$\overline{\text{CBACK}}$,$\overline{\text{CIIN}}$）は全てデータ・シートのセット・アップ時間，及びホールド時間を守る必要がある。

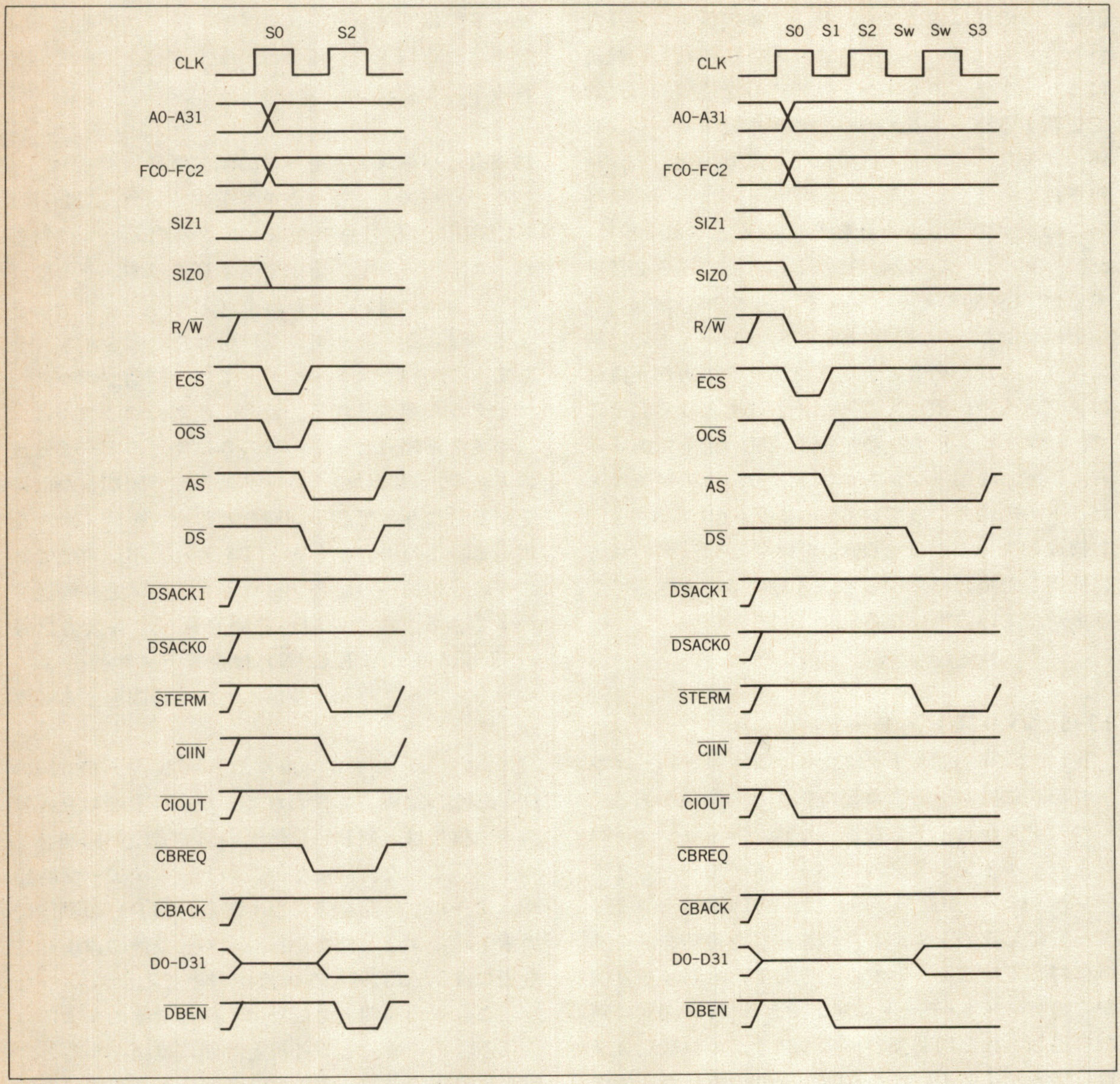
〔注〕：　従来の68000のDTACKや，68020のDSACK，又，両方で使われていたBERRやＶＰＡ等の入力信号は，ＭＰＵ自体が内部で完全に同期をとって用いていた。従って，このセット・アップ時間や，ホールド時間を守らなくても，ＭＰＵの動作に異常を起こすことはなかった。もし条件に違反しても更に１WAITが入るか否かの差でしかなかったが，これらの同期入力信号は，内部でそのまま用いられる。そこで，もし各規定時間に違反をすると，ＭＰＵ自体の動作，特にバスのアクセス機構に異常をきたすので厳重に注意を払う必要がある。

この同期型のサイクルは，通常のメモリ・アクセスでもバス幅が32ビットであれば用いる事ができるが，高速のメモリでないと無意味である。

バースト・モードは，これを満たす為には，幾つかの条件を必要とする。それは，

● 命令／データの各々のバースト・イネーブル・ビ

図13-5-1　同期モード・バス・サイクル・タイミングの例



ットがセットされている事（ＩＢＥ，ＤＢＥ）

- キャッシュ自体がイネーブルである事
- キャッシュがフリーズになっていない事
- ＲＭＣサイクルのリード時でない事
- ＭＭＵ内の相当するＡＴＣのＣＩビットがネゲイトされている事
- 二つのキャッシュ・タグに渡るアクセスでない事

以上の条件が満たされて始めてＭＰＵはバースト・モードを要求してくる。これは$\overline{\text{CBREQ}}$のアサートで確認される。外部のメモリ・ハードウェアはこのバースト・モードに対応する為の条件が別に存在する。

- バスが32ビットである事
- Ａ２，Ａ３のみのインクリメントのできる事
- 同期型バス・サイクルをサポートできる事

以上の条件が満たせる時に，外部のハードウェア

は$\overline{\mathrm{CBACK}}$をアサートしなければならない。この$\overline{\mathrm{CBACK}}$のアサート，$\overline{\mathrm{STERM}}$で終わるメモリ・サイクル，そして，$\overline{\mathrm{CIIN}}$や$\overline{\mathrm{BERR}}$のアサートのない事で，ＭＰＵはバースト・モードでの動作を始める。又，もしメモリ・サイクルを$\overline{\mathrm{STERM}}$でなく，$\overline{\mathrm{DSACKx}}$で終了させると，$\overline{\mathrm{CBACK}}$は無視され，バースト・モードには入らない。

一度バースト・モードの条件が満たされると，それは次のサイクルもバースト・モードを連続するという意味を持つ。従って，連続した４回のロング・ワードのリード・サイクルが実行されるが，その途中での$\overline{\mathrm{CIIN}}$のアサートや，$\overline{\mathrm{DSACKx}}$によるバス・サイクルのターミネイションは以後のバースト・モードを行なわせず，該当するエントリは全てインバリッドとなって使われない。

また，特記すべきは，この２，３，４回目のバースト・モード・メモリ・サイクルが$\overline{\mathrm{BERR}}$信号で終了させられた場合，当該エントリはノンバリッドとなり，その他には何の例外処置も行なわれないという事である。

又，同様にリトライの動作に対しては，第１回目のバースト・モードのスタートに対しては有効であるが，２回目以降のサイクルにおいてはリトライは起こらず，HALT,BERRの各々の動作を行なう事になる（まずBERRでバスの動作のみ停止して，メモリ・サイクル終了後にHALTとなる）。

13-6　キャッシュ自体の構造

68030のキャッシュ・メモリは，１ＴＡＧ，４エントリで，各エントリごとにＶフラグがあり，エントリの有効／無効を示している。この４エントリは，ＭＰＵのアドレス，Ａ２，Ａ３の組合わせで示される連続アドレスである。

タグは，命令用とデータ用で若干，差がある。それは，命令用はＦＣ２のみがタグに含まれているがデータ用では全てのＦＣコードが含まれる。

■データ・キャッシュとライト・アロケイト

68020にはなかったデータ・キャッシュは，命令用のキャッシュとは動作が若干異なる。命令の場合には，外部メモリから読出されたコードは，キャッシュが動作していれば，そのままキャッシュに入り，次に外部メモリ・サイクルの替りにこのキャッシュの内容が読出された。この読出しについては，データ・キャッシュも全く同じである。但し，データ・キャッシュはオペランドとしてアクセスされるものであれば，ＣＰＵ空間のアクセス以外の全てのメモリ・サイクルをキャッシュに入れようとする。これは，そのアクセスがＰＣ相対であってもMOVES命令による他の空間のものであってもキャッシュに入れる事ができる。

図13-6-1　ライト・アロケイトの機能

	キャッシュHIT	データ・アップデイト
WA＝1	HIT	○
	NON HIT	○
WA＝0	HIT	○
	NON HIT	×

ＭＰＵが書込みサイクルを起こす時に，データ・キャッシュとの関係は，場合により変化する。

ＭＰＵが，現在キャッシュの内にないアドレスへ書込みを起こした時に，この書込みをキャッシュに対して行なうか否かを決める制御ビットがある。これにより，ＭＰＵの書込み全てをキャッシュに入れてしまう場合と，キャッシュ内のタグに書込みアドレスが存在した場合のみ，キャッシュの内容をアップデイトする場合とがある。ここで書込みサイクルは必ず外部メモリ・サイクルを起こすので，MMUによるＣＩビット，外部からの$\overline{\mathrm{CIIN}}$信号への入力によっての制御は有効で，これらの場合，つまりＩ/Ｏへの書込み等は，当然，キャッシュに入らない。そうなると，この書込みサイクルをキャッシュがＨＩＴしない場合に行なうか否かは，どちらであっても構わない。この違いはソフトウェアの内容の差によるものとなる。まず書込みがあって，それからそのデータを使う場合には，キャッシュに書いて有効であるが通常の読んでから変更する様な場合には，書込みのみのアドレスをキャッシュに入れる事は禁止しておいて良い。この制御をどちらにするかは，そのシステムの特性によって決めるべきであろう。

■ライト・サイクルのバス・エラー

キャッシュに入るべきライトのメモリ・サイクルで外部のメモリ・サイクルでバス・エラーが発生する事もあり得る。この時に，当然外部には，このライト・データは保存されないが，キャッシュには入ってしまう。そこで，MMUによって読出しのみに割当てていたエリアに書込みを行ない，そこの内容がキャッシュに入っていた場合には，そのキャッシュの内容は失われ，バス・エラーの例外処理が発生する。

13-7 バスの動作

68030のバスの動作は，68020のバスの動作の拡張版ある。68020のバスの動作は全てサポートされている。但し，一つだけ例外があって，ダイナミック・バス・サイジングによってバスの幅を変えられる境界は必ずロング・ワードの整合を必要とする。これ以外はバスの従来からの使い方は同じである。

これは，バス・アクセスに対するクロック・サイクルの数，位相関係まで，同じであり，内部にMMUを持っても，このMMUの変換時間は完全にMPUの並列動作に組込まれて，余分のクロック数をMMUの為に必要とはしない。

68030のバス動作で最も異なるのは，このMPUがフル32ビット・オリエンテッドになっている証しでもあるが，同期型のバス・サイクルをサポートした事であろう。

この同期型のバス・サイクルとは，単純に言えば32ビット幅のバスの場合，アクノレッジとしてDSACKxを用いる代りにSTERMという信号を用いて，メモリ・サイクルを終了させる方法である，という事ができる。

従来，アクセスされたバスの幅をDSACKxによってアクノレッジとして返していたのと同様に，バースト・モード以外ではメモリ・サイクルのスタート，MPU側の出してくる信号，並びにタイミングは全く従来のアクセスと同じである。

そこで，アクセスされたメモリが，高速のメモリであれば，この同期型サイクルとしてのアクノレッジを返す事によってメモリ・サイクルの時間を短くできる。

この場合，最速2クロックで1メモリ・サイクルが終了する。もちろん，読出し，書込みとも同じである。

13-7-1 タイミング

同期型サイクルでは，外部から与える信号，$\overline{\text{STERM}}$, $\overline{\text{CIIN}}$, $\overline{\text{CBACK}}$は，MPUのクロックに外側で同期してS2の立ち上りに対して定められた，セットアップとホールド時間を守って与えられなければならない。これは，従来DSACKやBERR，AVEC等を与えていたのと異なり，もしこの同期に失敗すると正しい動作が保証されないという点で異なる。従来の非同期系の信号にもセットアップとホールドの時間規定はあるが，これは内部での同期回路の為のもので，違反しても別に誤動作とはならずメモリのサイクル時間がバラついたり，反応が遅れたりするだけであった。

しかし，同期型バス・サイクル時には，MPUのバス・アクセス機構自体の動作が，もし規定に違反すると，正しく行なえず，MPUが正しい実行もできなくなる。そこで，これらの信号は必ず同期したものを用いねばならない。

この同期型サイクルの動作も，従来の68000ファミリと同様に半クロックを1ステートとして，規定されている(図13-7-1参照)。

S0：ECS，場合により，OCSが出力される。アドレス，FCコード，R/W, SIZx,CIOUTが出力される。

S1：AS，DBENが出力される。ECS，OCSはネゲイトされる。

S2：STERMが認識される。もし必要なら，$\overline{\text{CIIN}}$，$\overline{\text{CBACK}}$も認識される。

S＊：S2でSTERMがアサートされない時に，1クロックのアイドル・サイクルが入る。そして，このアクセスが書込みなら，DSがアサートされる（従って，ノーウェイト時は，書込み時のDSは出力されない)。

S3：AS，DS，DBENをネゲイトする。ASがネゲイトされたら，STERMをネゲイトする。他の信号は，S3の終りまで有効となっている。

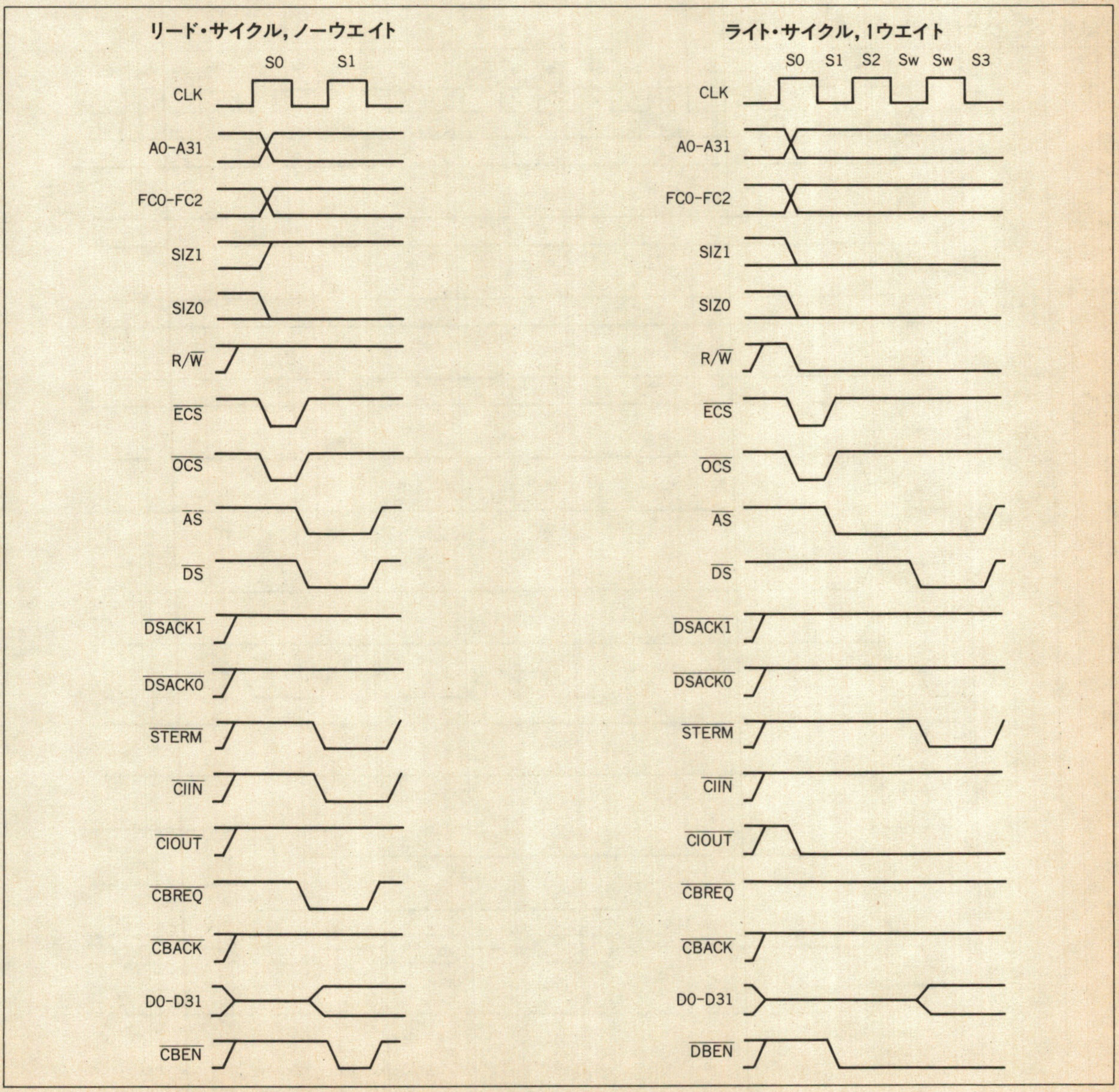
13-7-2　バースト・モード

バースト・モードはキャッシュの４エントリを満たす為に行われる動作で，この同期モードでのみサポートできる。バースト・モードでは，ＡＳと同時に$\overline{\mathrm{CBREQ}}$信号がアサートされる。これに対してバス・エラー，$\overline{\mathrm{CIIN}}$のアサート，$\overline{\mathrm{CBREQ}}$のネゲイトによりバースト・モードは中断される。但し，これらの信号は現在のメモリ・サイクルでなく，次のサイクルの中断を示している。

メモリ側がバースト・モードをサポートできる時は，CBREQに対しCBACKをアサートして，バースト・モードを受け入れる事を宣言する。この信号のある限り，DSACKxは無視される。

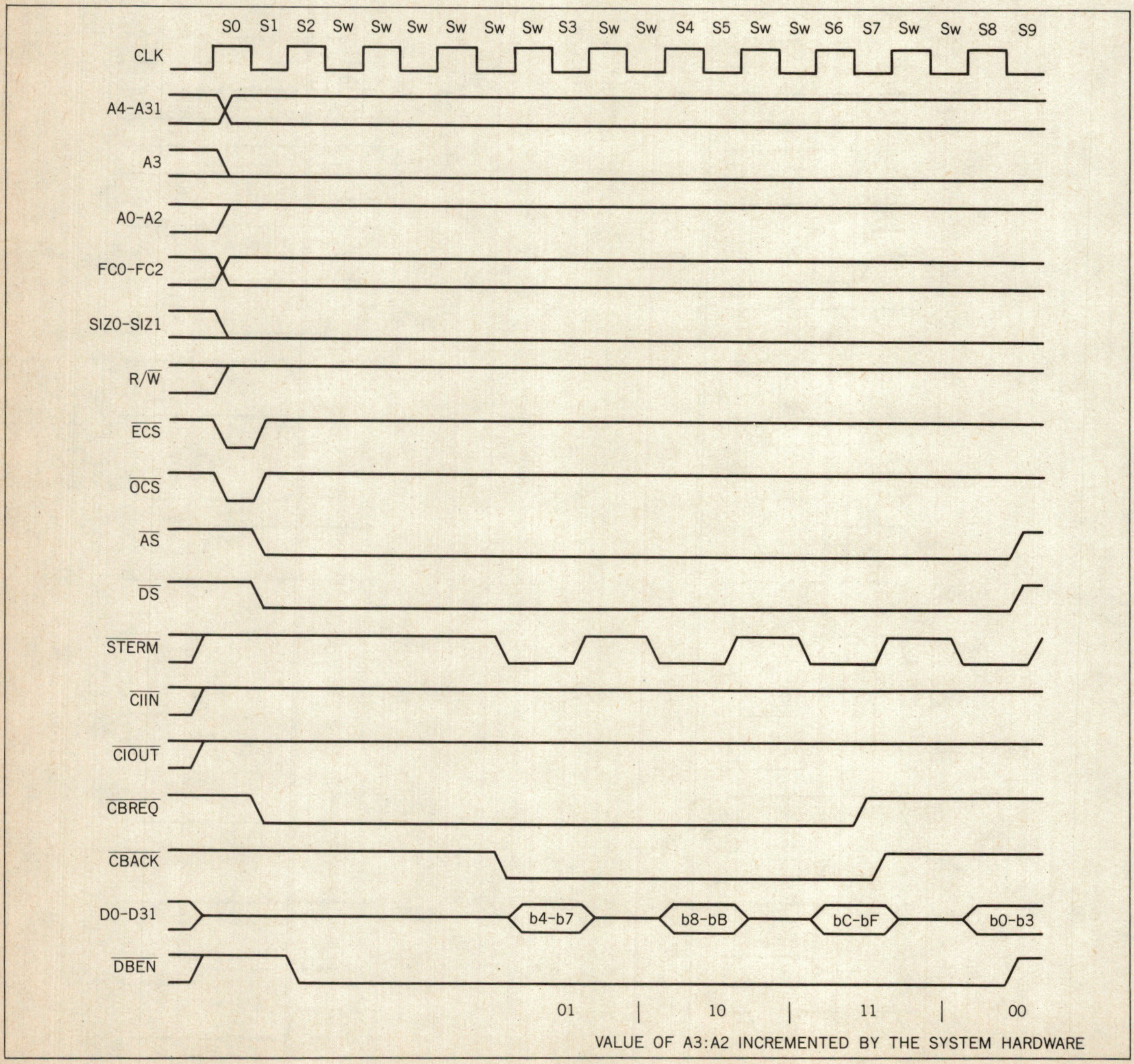
そして，Ｓ２，又は更に待ち時間が入っての等価的Ｓ２の立上りでSTERMを認識するとＳ２の立下

図13-7-1　68030同期型バス・サイクル・タイミングとステート数



り（Ｓ３のスタート）で，バス上のデータをＭＰＵを取り入れる。このＳ３の終了時には，STERMはネゲイトされ，次のデータへのアクセスに入る。この時ＭＰＵの出力信号は何の変化もない。外側で，ＭＰＵのクロックに同期して，STERMとデータをＭＰＵに与えていく。この時，四つのエントリに与えるデータは，アクセス・アドレスから始まって，Ａ２，Ａ３で示される。4ロング・ワード分となる。３回のアクセスが終わるとCBREQはネゲイトされて，４回目のアクセス以降がない事を示す。又，もしこの間に$\overline{\text{CIIN}}$のアサート，$\overline{\text{BERR}}$のアサートがあると，そのメモリ・サイクルでバースト・モードは打ち切られる。

図13-7-2　バースト・モードのタイミング例



第14章

68000/10システムの68020への変更
68020システムの68030への変更

14

14-1 ハードウェアの全体構成

68020は高機能であるとともにハードウェア，特に外部のバス・ラインに対しては融通性に富んだ構造を持っている。そこで，現行の68000や68010のシステムに68020を接続して容易に68020のシステムに変更してしまう事が可能である。

以下にその実例を示す。もちろんこの場合，68020の持つ全32ビットのアドレス範囲が使える訳ではないし，性能的には大幅に落としめる事になるが，簡単であるという点が良く，ここから拡張して本格的なシステムに進まれるのが良いと思われる。この時，本書内で指摘した種々のハードウェア上の留意点には特に注意を払っていただきたい。

この68000アダプタはMPUとして68020，コ・プロセッサとして68881，デコーダ及びタイミング制御は3個のプログラマブル・アレイ・ロジック素子（PAL)を使っている。そこで68000／10のICソケットに挿入可能なプラグを設けて，このプラグを通して68000／10としてのMPUのハードウェア機能を68020が提供する形をとっている。

システム内には通常のメモリ，RAMやROMは既に存在しているはずであり，入出力も同様であるので，このアダプタ内には，その様なメモリやI／Oはなく，単純極まりないものとなっている。

2個のデータ・バス・バッファは内部の68881との通信中，外部をアクセスしないためと，外部の負荷に対して余裕をとるために使用されている。

アダプタ内のデータ・バス幅は32ビットであり，68881とは32ビットで通信している。もし，将来，システムを拡張して高速化を図るなら，この内部の32ビット・バスに高速のメモリを接続するのがよいがその時，この構成のままだと外部から内部へDMA転送ができないし，16ビットのバス・バッファも32ビットに拡張する等，種々の変更が必要となる。

14-2 システムの性能

このアダプタを用いてもシステム全体の性能の向上は少しである。その最大の理由はメモリの速度及びクロック周波数である。特に68000の8MHz用に作られているシステムのメモリ速度では68020は50%以上停止していると言える。

このアダプタは現行のシステムに少しの変更で68020を接続し，FPCPや，68020の拡張された命令やアドレス・モードを実験，学習する目的を持つもので68000ベースの68020クロス・アセンブラを持っているシステムでは，そのクロス・アセンブラがクロスでなく，レジデントなアセンブラとなり，実際に実行できる事である。

更に，メモリ管理を持っていないシステムではコンテクスト・スイッチによるキャッシュのミス・ヒットは行なわないので，キャッシュ・メモリを使って各ソフトウェアに対するキャッシュの有効度を確認できる。ただしこれには，MPU内のレジスタの操作が必要なので，完全に68000のソフトウェアのままで実行するわけにはいかない。

14-3 ソフトウェアの互換性

もし，多くを望まないなら，68000と68010／20との例外処理範囲の差異(MOVE from SR，12章を参照）を除けば68000のオペ・コードはそのまま68020で実行できる。そこで，現行68000／10のソフトウェアはそのままでこのシステム動作できる。

例外処理のベクタの位置や，ベース・ページの範囲も同じであるから，この点では何ら問題はない。

もしソフトウェア上で問題があるとすると，68000と68020の共用のレジスタSRで，68000では未使用だったビットで68020が使用している所（例えば，M，T2）を68000のソフトウェアが誤って操作している場合がある。このビットは従来は何の機能も持っていなかったので，バグにも何にもならなかったが，もし68020で動作させれば，もちろん，バグとなる。

これ以外には68020で使われ始め，68000では，未使用，ただし予約済となっている例外処理ベクタを何かに使っている場合である。

これは68020の拡張された部分を使わない限り使

われないが，ＭＰＵとして，使いだす可能性があるので，もしこういう点があれば予め変更をしておく必要がある。

ソフトウェア上での更なる問題点は，68020のアドレス・モードの拡張によるＯＰコードの変化である。これについてはユーザーズ・マニュアルの付録に例が掲載されているが，いくつかの命令コードの意味する命令内容が68020は68000／10と変わっている。ただ，こうした命令は数少ないし，特別な場合に限られるので，一般には問題とならない。

14-4　ハードウェア上の問題点

このアダプタは基本的には68000のバス信号を68020より作り出しているが，ＭＰＵの基本的な性質の違いのために問題がないとは言えない。特に８ビット系の低速度の素子がＩ／Ｏによって使用されている時にキャッシュを生かすと，Ｉ／Ｏへのアクセスが連続して生ずる。この連続の度合いがＩ／Ｏディバイスの許容限度を越すと（ほとんどのディバイスが該当する），ディバイス側でエラーを生ずる。このため複数のＮＯＰ命令をＩ／Ｏの操作ごとに入れる必要が生じる場合が多い。

また，もし外部のシステム側で，クロックの幅を操作していると，バスのタイミング・スペックに違反する恐れがある。この事は，現行の68020-12MHzバージョンの動作最低周波数が６MHzであるので，クロックの幅に細工をされるとＭＰＵの動作不良を引き起こす可能性も高い。このアダプタを入れて実験するシステムではクロックは10または12.5MHzである事が望ましい。

14-5　個別機能の説明

14-5-1　68020

ＭＰＵはアダプタ内では32ビットのデータ・バスを使いFPCPをサポートしている。この場合の特例はA24～A31の上位のアドレス線が未使用になっている事のみである。ソフトウェアの実験に便利な様に，ＭＰＵの外部よりキャッシュの使用を禁じたり許可したりする入力ピンCDISについてはジャンパを設けて，コントロールできるようにした。

14-5-2　68881（FPCP）

FPCPは最も理想的な形でＭＰＵに接続されている。データ・バス・サイズは32ビット＝ロング・ワード固定なので，ＳＩＺとＡ０のピンは固定されている。

14-5-3　Ｕ１内部デコーダ（ＰＡＬ）

これによって，大きく分けて二つの機能を生じさせている。一つは，ＳＩＺコードとＡ０，ＤＳより68000のＡＳ，ＬＤＳ，ＵＤＳを作っている。

デコーダとしてはFPCPへのアクセスをＦＣ０，ＦＣ１とA16，A17より作り出して，FPCPに与えている。この他にも割込みアクノレッジ・サイクルのデコード，データ・バス・バッファの制御も行なっている。

14-5-4　Ｕ２同期型バス・サイクル発生器（ＰＡＬ）

これにより，68020のサポートしていない同期型のＩ／Ｏのサポートを行なっている。内部のＦ／Ｆにより，Ｅクロックをシステム・クロックより発生しこれと同期するＶＭＡも作っている。更に，割込アクノレッジ・サイクルにはＶＰＡをAVECに置換える作業も行なって，同期型８ビットの素子からの割込みも，サポートしている。

14-5-5　Ｕ３ＴＡＳ命令サポート（ＰＡＬ）

これはマルチＣＰＵのシステムで，共有メモリによるフラグのやりとりでも行なっていない限り，つまりＴＡＳ命令を使っていない限り不必要なものである。68020ではＲＭＣが連続するメモリ・サイクルを保障するが，68000ではＡＳが連続して同じ機能を果たす。このＲＭＣをＡＳに置換えるためのロジックである。

図14-5-1　FPCPの専用空間のデコード68020のAS, DS, SIZEを68000のAS, UDS, LDSへの変換
内部アクセスと外部アクセスの差による，データ・バス・バッファの制御

```
ABEL(tm) Version 1.13  -  Document Generator
logical address decoder with 68010 emulator ver 2.0
Equations for Module emu10_lad

Device em10u1


     Reduced Equations:

       fpcpsel = !(!a16 & a17 & fc0 & fc1);

       enable pas = (bgack);

       pas = !(!aslock
             # a16 & !as
             # !a17 & !as
             # !as & !fc0
             # !as & !fc1);


       enable lds = (bgack);

       lds = !(a0 & !ds # !ds & !siz0 # !ds & siz1);

       enable uds = (bgack);

       uds = !(!a0 & !ds);

       pdben = !(a16 & bgack & !dben
               # !a17 & bgack & !dben
               # bgack & !dben & !fc0
               # bgack & !dben & !fc1);


       IackCycle = !(a16 & a17 & bgack & fc0 & fc1);

                            P16L8

                 ----------¥     /----------
                 |          ¥   /          |
                 |           -----         |
        bgack    |  1                  20  | Vcc
                 |                         |
           a0    |  2                  19  | IackCycle
                 |                         |
         siz0    |  3                  18  | lds
                 |                         |
         siz1    |  4                  17  | uds
                 |                         |
          a16    |  5                  16  | ds
                 |                         |
          a17    |  6                  15  | pas
                 |                         |
       aslock    |  7                  14  | fpcpsel
                 |                         |
          fc0    |  8                  13  | dben
                 |                         |
          fc1    |  9                  12  | pdben
                 |                         |
          GND    | 10                  11  | as
                 |                         |
                 |                         |
                 ---------------------------
```

図14-5-2 VPAによる,VMA信号,E信号の発生

```
ABEL(tm) Version 1.13  -  Document Generator
6800 protocol interface
Equations for Module emu10_6800IF

Device em10u2

     Reduced Equations:

       enbl := !(!reset # !enbl & !q0 # q0 & !q1 # !q2);

       q2 := !(!reset # enbl & !q2 # !q1 # q0 & !q2);

       q1 := !(!reset # !q0 & !q1 # enbl & !q0 # !q1 & q2 # enbl & !q2);

       q0 := !(!reset # !q1 & q2 # q1 & !q2 # enbl);

       SynedVpa := !(!vpa);

       avec = !(!as & !iackcycle & !vpa);

       dsack1 = !(!as & !dtack
                # !as & !fpcpack
                # !as & enbl & !q0 & !q1 & !vma);


       vma := !(!SynedVpa & !enbl & !q0 & q1 & !q2
              # enbl & !q0 & !q1 & !vma
              # q1 & q2 & !vma);

                              P16R6

                    ----------¥       /----------
                    |          ¥     /          |
                    |           -----           |
              clk   |  1                    20  |  Vcc
                    |                           |
              vpa   |  2                    19  |  dsack1
                    |                           |
            dtack   |  3                    18  |  enbl
                    |                           |
          fpcpack   |  4                    17  |  q2
                    |                           |
                    |  5                    16  |  q1
                    |                           |
                    |  6                    15  |  q0
                    |                           |
               as   |  7                    14  |  SynedVpa
                    |                           |
            reset   |  8                    13  |  vma
                    |                           |
        iackcycle   |  9                    12  |  avec
                    |                           |
              GND   | 10                    11  |  oe
                    |                           |
                    |                           |
                    -----------------------------


  end of module emu10_6800IF
```

図14-5-3　このPALはOPTIONで，マルチCPUを行なう場合のTAS命令のサポートを行なう場合のみに用いる

```
ABEL(tm) Version 1.13  -  Document Generator
read modify write cycle support
Equations for Module emu10_aslock

Device em10u 3

     Reduced Equations:

       asb = !(as);

       rmc_a := !(!rmc);

       rmc_b := !(!rmc_a);

       aslock = !(!rmc_a & rmc_b);

                         P16R4

                  ----------¥      /----------
                  |          ¥    /          |
                  |           -----          |
              clk |  1                  20   | Vcc
                  |                          |
               as |  2                  19   | asb
                  |                          |
              rmc |  3                  18   | aslock
                  |                          |
                  |  4                  17   | rmc_a
                  |                          |
                  |  5                  16   | rmc_b
                  |                          |
                  |  6                  15   |
                  |                          |
                  |  7                  14   |
                  |                          |
                  |  8                  13   |
                  |                          |
                  |  9                  12   |
                  |                          |
              GND | 10                  11   | oe
                  |                          |
                  |                          |
                  ----------------------------

end of module emu10_aslock
```

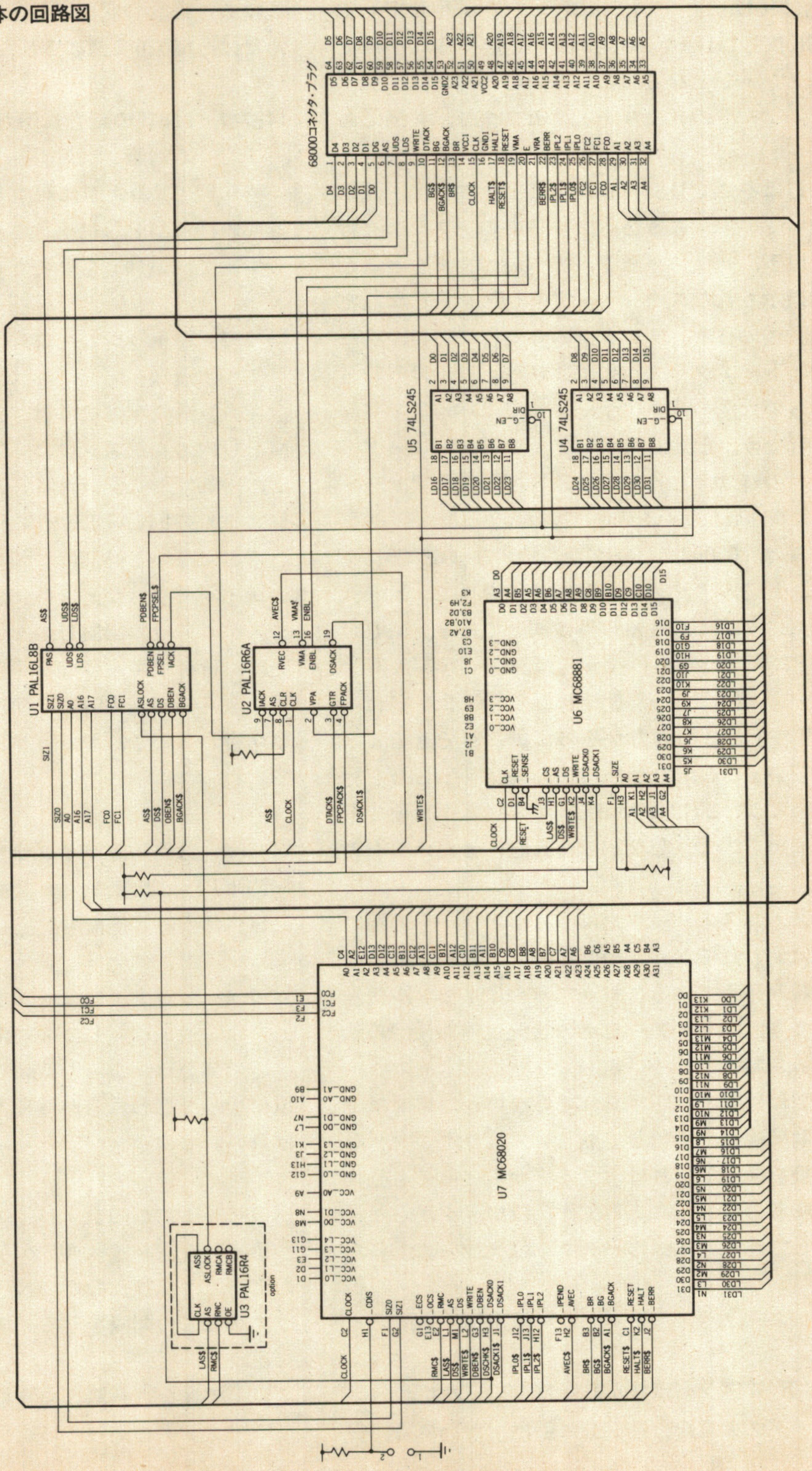
14-5-6　U 5，6 データ・バス・バッファ

このバッファは68020のファン・アウトでは配線長の伸びのためにデータ・バスのタイミングが悪化するのを防ぐと共に，FPCPとMPUの通信時，外部と切り離し，更に外部よりDMA等のバス要求に正しく答えるためのものとなっている。

14-6　回路上の説明

このシステムは電源が投入され，リセット信号が与えられると，それはそのままMPU，FPCPに与えられる。リセットが解除されると，MPUは，0番地へロング・ワードのアクセスを行ない，ベクタを取りにいく。この時，68000用のプラグのデータ・バ

図14-5-3　システム全体の回路図



スの幅は16ビットなので，プラグから返ってくるDTACK信号はDSACK１にＵ２で変換されワードのアクノレッジとなる（これは，この後のロング・ワード，ワードのメモリ・アクセス全て，共通である)。

ＭＰＵはそこでD31～D16より上位ワードを読取り，続いてアドレス＄２へ，ワードのアクセスを行なう。以下，同様の手順で，二つのベクタをとるとそこで指示されたスタート・アドレスをＰＣに設定して，動作に移る。プログラムの実行に伴い，データ・バイト・アクセスが要求されるとＵ１はＬＤＳかＵＤＳの片側のみをアサートして，バイト・アクセスを行なう。前述したロング・ワード，ワード・アクセスの時は，ＬＤＳ，ＵＤＳは両方共アサートされて，68000と互換をとっている。

割込みは，68000のプラグからそのまま68020に与えられ，割込みアクノレッジ・サイクルで，ＶＰＡがAVECに置換えられる以外は同一である。ＭＰＵがFPCPにアクセスを行なう時は，ワード幅も，ロング・ワード幅もあり得る。そこでFPCPからは，DSACKの１と２の両方がそのままの形で返るようになっている。これのうち，DSACK１が，Ｕ２を経由しているが，これは，ＭＰＵのDSACK１は他でも使われているので，Ｕ２内でＯＲをとるためのものである。

データ・バスのバッファはバイト・ワードの幅に無関係に使われており，ＭＰＵのＲ／Ｗ信号により方向を切り替えている。バッファのイネーブル信号はＵ１より，バス・アービタの状況と内部使用の状況によって制御されている。

回路図には電源は明示されていないが，電源はできれば別配線にて供給して欲しい。プラグが充分な電流容量を持っていれば，プラグからとっても動作する（筆者の手もとにある試作品はプラグからとって動作している)。電源のパスコンは充分入れていただきたい。

14-7　アダプタに対応するソフトウェア

このアダプタの能力を充分に引出すためには，ＯＳにある程度の変更を与える必要がある。そのほとんどは，68020の追加された機能のためである。

14-7-1　68020の新たに追加された例外処理に対する対策

特にFPCPの使用に対するものと，フォーマット違反の様なＭＰＵに追加された機能に対するものは必ず変更，追加していただきたい。

14-7-2　キャッシュ・コントロール・プログラムの追加

キャッシュ・メモリは，外部のピンのみでなく，ＭＰＵ内のレジスタによっても制御されているのでこの内部レジスタ（キャッシュ・コントロール・レジスタ）のENABLEビットを１にして使用を許可しておかないと外部のジャンパを制御してもキャッシュ・メモリは使えない。またMMUを持っているシステムでは，MMUのセグメント・アサインの変更時には必ずキャッシュをクリヤする必要もある。更に例外処理発生時も，MMUのアサインは自動的に切替わるので，キャッシュをクリヤする必要がある。

14-7-3　数値演算ライブラリの入替え

FPCP用に計算ルーチンを入れ替える。この時ＭＰＵ自体の計算機能も向上している点に注意がいる。

14-7-4　ビット・フィールド，アドレス・モードの追加に対しての対応

この部分は，まったく新しいソフトウェアを作る事になると思われるが処理の必要度に応じて行なえば良い。

14-7-5　Ｉ／Ｏのディバイス・エラーの対策

Ｉ／Ｏディバイス側でエラーを生ずる時は，１回のアクセス直後に数個のＮＯＰを入れて，アクセスとアクセスの間隔をとる必要がある。

以上の様な点を処理する事により，かなり高度にこのアダプタを使っていただけるものと思う。

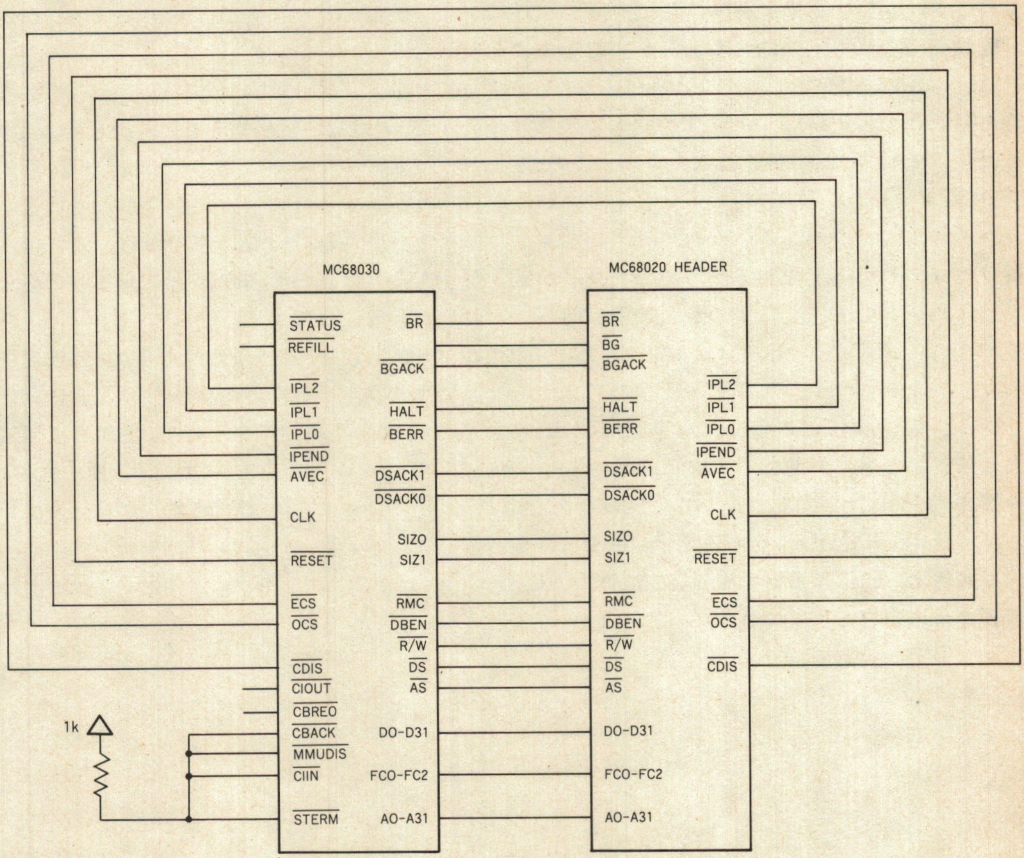
14-8　68030の68020システムへの適用

この項は，68030ユーザーズ・マニュアルの附録にある68030を現行の68020のシステムで動作させてみる為の適用ノートの紹介と，解説である。

まず，現在の68020用ＩＣソケットに入るべき68030との変換接続器の内容は，**図14-8-1**に示されている。68000ソケットに68020を入れる場合と異なり何の部品(抵抗１本を除き)も不要である。これは，68030は68020のバスの機能の全てを有しており，ダイナミック・バス・サイジング他も全く同じであり更にデータ／アドレスの幅も同じなので，何の問題もないからである。

必ず行なうべき処理としては，68030だけが有する高速のクロック同期バス・ハンドシェイクを禁止しておく事である。この為に，STERMとCBACKはHにプルアップしておかねばならない。又，この接続の意義の第一はMMUの評価及びソフトウェアの作成であるから，$\overline{\text{MMUDIS}}$もHにして，MMUはイネーブルできる様にしておく。**図14-8-1**の例では，$\overline{\text{CIIN}}$もHにプル・アップされているが，これは，必ずしも正しいとは限らない。MMUを用いずに，データ・キャッシュをイネーブルにするとＩ／Ｏアク

図14-8-1　68020-68030変換接続器



セスまでキャッシュが応答してしまうからである。ここをこのままプル・アップしておく時には，まずMMUを生かして，各ATC内のCIビットの操作により，キャッシュに取り込まれては困る部分に相当するエントリに禁止を指示しておいてからにする必要がある。そうでないならば，I／O空間等のアクセスで，この$\overline{\text{CIIN}}$をアサートする様にしなければならない。

次にCIOUTは，もし外部にデータ・キャッシュがあるのなら，その禁止信号にORがとられなければならない。

CBREQは，たびたび出力されてくるが，これは無視するので何もせず，N.C.とする。

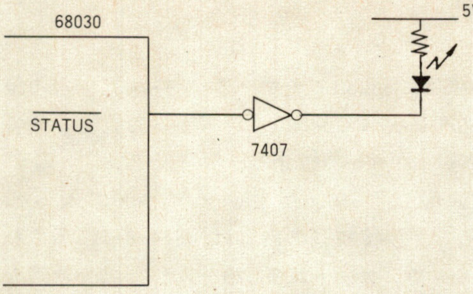
STATUS，REFILEの２信号は，MPUの状況を示すので，何もしなくても良い。但し，STATUSには，**図14-8-2**の様な回路にてLEDを点灯する事を勧める。これは68030では，ダブル・バス・フォールト時のMPUの停止時に，HALTが内部よりアサートされず（HALTは68030では入力専用）このSTATUSにダブル・バス・フォールトによるシステムの停止が示されるからである。

信号ピンの差は以上であるが，この変換器が働く為には，ハードウェア上に幾つかの条件がある。

その１は，PMMU68851のない事である。68030は，コ・プロセッサID＝0を外部へのアクセスはできない。もし68030の内部MMUを$\overline{\text{MMUDIS}}$入力により禁止しても，外部へのアクセスはできないから外部のPMMUは必ず取り除かなければならない。

これに伴い，PMMUに対する命令，モジュール・コール関係と，BKPT命令のセット・アップ，PCSRへのアクセス命令も実行できない。

図14-8-2　ダブル・バス・フォールト用LED点灯回路



先にも述べたが，アクセス・レベルの機構のない68030ではCALLM，RTMの２命令は削られている。同様にCRPに連結して動作していたタスク・エリアも除去されているので，コンカレント・スイッチごとに全てのATCをフラッシュする必要も生じる。

以上はPMMUの除去に伴う変更要求点であったが，PMMUを元々持たないシステムでも，場合により若干の問題が生じる。それは13章68030項で説明したがロング・ワードに整合されていないバス幅の変更が許されていない点である。これは，まれな例であるとは思われるが，もしあるとアクセスできないので要注意である。

又，もし，FPCPが元の68020に接続されていた場合には，ほとんどの場合，そのまま動作する。もし68030のクロックを上げて，性能を上げようとする時のみ，FPCPの方もクロックの速いものとするか，別のクロックのまま，デコーダを見直すかを判断すれば良い。

参考文献

(1). MC68020ユーザーズ・マニュアル（和文）

(2). 〃 （英文）

(3). MC68000マイクロプロセッサ・ユーザーズ・マニュアル (4)和文

(4). 〃 英文

(5). Design Philosophy Behind Motorola's MC68000 (AR208) PART 1

(6). 〃 PART 2

(7). 〃 PART 3

(8). MULTIPROCESSING CAPABILITIES OF THE MC68020 32-BIT MICROPROCESSOR (AR220)

(9). 68000リアルタイム・マルチタスキング・ソフトウェア・ユーザーズ・ガイド

(10). Virtual Memory Using the MC68000 and the MC68451 MMU (DC-001)

(11). 68000 Data sheet

(12). 68008 Data sheet

(13). 68010 Data sheet

(14). MC68020 Microprocessor to VERSA bus/VME bus Interconnection Techniques

(15). MC68000 Article Reprints

(16). MTT12, MTT15モトローラ(VERSA-DOS)

(17). MC68000 Family Structured MACRO Assembler manual

(18). 68451Data Sheet

(19). 68851User's Manual

(20). 68851Technical Summary

(21). 68881User's Manual

(22). MC68030ユーザーズ・マニュアル（英文）

筆者略歴

たつの　いさお
辰野　功

1950年　4月26日　「戦争を知らない子供達」の世代として生まれる。
1966年　4月　早稲田高校入学。同年12月、電話級アマチュア無線技士の資格を得て、真空管の洗礼を受ける。
1970年　4月　一浪ののち、東京理科大、芝浦工大、慶応いずれも不合格の苦汁を味わうものの、あこがれの電気通信大学材料科学科入学。
1972年　9月　大学3年在学中に結婚。現在2児の父。
1973年　MC6800とめぐり合い。それ以来、一貫したモトローラ・ファン。
1974年　4月　㈱デジテックス研究所就職。アルバイトからシフトしたため、リクルートの経験、無し。
1979年　㈱アムスク入社(旧丸栄商事)。現在、技術担当取締役。
1980年　パチンコの777フィーバー機の設計で大ヒットを飛ばす。

生活状況：1年の半分が単身赴任。
趣　　味：作文。
スポーツ：剣道2段。3段に挑戦中。現在初段の長男の猛追を受けている。
酒　　　：もっぱらバーボン派。「日本酒は原酒にかぎる」が口ぐせ。
特　　技：英会話(通訳可)。毎週水曜日には英会話教室に、せっせと通う。
回路、ソフトのバグ発見(心理学の応用による?!)

検	印
省	略

実践的システム設計のための
68020ファミリ活用の実際 改訂版

© 1989 Dempa publication, Inc.

1989年5月8日　第1版　第1刷　定価3090円(本体3000円)

著　者　辰野　功
発行人　平山秀雄
発行所　㈱電波新聞社
〒141　東京都品川区東五反田1-11-15
電話　03(445)6111(大代表)
振替　東京5-51961
印　刷　大日本印刷㈱
製　本　㈱堅省堂

※落丁，乱丁はお取り替えいたします。　ISBN4-88554-202-2 C3055 P3090E

実践的システム設計のための

68020ファミリ活用の実際

第１章　高級マイクロプロセッサへの進展と、その位置付け

第２章　モトローラ68000ファミリの開発思想

第３章　MC68000シリーズの特徴

第４章　一般的OSの使命と、その役割り

第５章　68000ファミリにおけるOSのサポート

第６章　68000ファミリの機能内容

第７章　命令解説

第８章　68000ファミリのアセンブラ書式

第９章　メモリおよびハードウェア

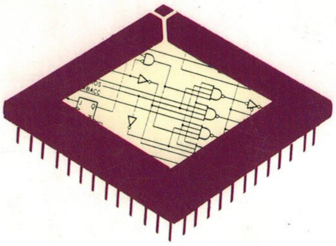
第10章　割込み処理

第11章　コ・プロセッサ

第12章　MMUの役割と機能解説

第13章　MC68030の機能解説

第14章　68000/10システムの68020への変更
68020システムの68030への変更



ISBN4-88554-202-2　C3055　P3090E　定価3090円（本体3000円）

Computer Design

実践的システム設計のための68020ファミリ活用の実際

辰野　功　著

電波新聞社

実践的システム設計のための——改訂版

68020ファミリ活用の実際

68030、68020、68010、68000、68008の機能詳解

辰野 功 著

電波新聞社

ISBN4-88554-202-2 C3055 P3090E 定価3090円(本体3000円)

売上カード

定価 三、〇九〇円（本体 三、〇〇〇円）

書名 改訂 68020ファミリ活用の実際

㈱電波新聞社

補充注文カード

書店名

注文数　　冊

発行所 ㈱電波新聞社

改訂 68020ファミリ活用の実際

ISBN4-88554-202-2 C3055 P3090E

4943

定価 3,090円
（本体 3,000円）

◘売れた商品です。至急補充して下さい。
◘店頭切れはお店の損です。

補充月日

12	
11	
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	

売り上げの都度，必ず帖合へ回して補充して下さい。